Jaihoe

Documentation
First Edition (version 1.2.5)

Developed by
Jerwin Keith Dela Cruz
jerwinkdc@outlook.ph

CONTENTS

INTRODUCTION **11**
Getting started with Jaihoe Blocks **12**
Block Foundation **14**
Block Structure **14**
Block Directive Distinction **15**
Block Patterns **16**
Importing Dependencies **17**
External Block Foundation **17**
Errors **18**
Basic Syntax **19**
- Starters 19
  - Displaying Hello World (j-hw) [Easter Egg] 19
  - Display About Jaihoe (j-about) 19
  - Reload Page (j-reload) 20
- Variable 21
  - Using “var” variable (j-var) 21
  - Using “let” variable (j-let) 22
  - Using “const” variable (j-const) 23
  - Setting/changing variable (j-set) 24
  - Set/change block variables (j-set-block) 25
  - Variable Value (j-value) 26
- If-else Statement 26
  - Using if (j-if) 26
  - Using else if (j-else-if) 26
  - Using else (j-else) 26
  - All together (If-else statement) 27
  - Using if block (j-if-block) 28
  - Using else if block (j-else-if-block) 28
  - All together (If-else block statement) 28
- Switch Statement 29

- Foundation (j-switch) 29
- Case (j-case) 29
- Break (j-break) 29
- Default (j-default) 29
- All together (Switch Statement) 29

➢ Loops 30
- For Loop (j-for) 30
- For-In Loop (j-for data-in) 31
- For-Of Loop (j-for data-of) 31
- While Loop (j-while) 32
- Do While Loop (j-do) 33
- Continue and Break (j-continue …) 33

➢ Functions 34
- Making a function (j-fx) 34
- Calling a basic function (j-fx-call) 34
- Working Example 34

➢ Objects 35
- Foundation (j-object) 35
- Attributes (j-attr) 35
- All together (Objects) 36

➢ Stringify 37

➢ Return 38

➢ Output 39
- Direct Write and Append 39
- Element Write and Append 39
- Var Write and Append 39
- Element Block Write/Append 39
- Direct Block Write/Append 40
- Working Example (j-write & j-append) 40
- Omit Method Write and Append 41
- Working (j-write & j-append) No Method 41
- Var, Element, or Direct New Line (j-br) 42

➢ Document Selector 43

- Get Selectors (j-docget) 43
- Self Selectors 44
- Property Extend (data-docget-extend) 44
- Property Apply (apply="") 45
- Property Apply Action (apply="action") 46
- Property Apply Set (apply="set") 47

➢ Document Attributes 47
- Set Attribute 47
- Has Attribute 48
- Get Attribute 48

➢ Event Listener 49
- Element (j-add-event element) 49
- Var (j-add-event var) 49
- Event Single Function Call (j-fx-evt) 49
- All together (Event Listener) 49

➢ Math 51
- Properties 51
- Math Methods 51
  - **Single Value Methods** 51
  - Double Value Methods 52
  - Multi Value Methods 52

➢ Random Number 53
- Random Entry Value 53
- Single Entry Value 53
- Double Entry Value 54

➢ Random Sentence 54
- Sentence Generator 54
- Lorem Generation 55

➢ String 55
- Length 55
- Default Methods 56
- Search Methods 65
- Extended Methods 69

- Numbers 76
  - Number Methods 76
  - Properties 85
  - Extended Methods 87
- Array 93
  - Default Set 93
  - Separate Set 95
  - Array Methods 96
  - Extended Methods 121
- Sets 127
  - New Set 127
  - Methods 128
  - Property 132
- Maps 133
  - New Map 133
  - Methods 135
  - (For of) Methods 139
  - Property 142
- Classes 143
  - Foundation (j-class) 143
  - Constructor (j-class-cfx) 143
  - Class Function (j-class-fx) 143
  - All together (Preview) 143
  - All together (Code) 144
- TypeOf 144
  - Foundation 144
  - All together (Preview and Code) 145
- Time Methods 146
  - Set Timeout 146
  - Clear Timeout 147
  - Set Interval 149
  - Clear Interval 150
- Promise 151

- Foundation 151
- Working Example 152

➢ Async and Await 153
- Foundation 153
- Working Example 154

➢ Fetch 155
- Foundation 155
- Working Example (Default) 156
- Working Example (Async and Await) 157

➢ Navigator 158
- Methods 158
- Altogether (Navigator) 159

➢ Location 160
- Methods 160
- Property 163

➢ Error 167
- Try and Catch 167
- Throw Statement 168
- Finally Statement 169

➢ Alert 171
- Legacy 171
- Alert Notification 171
- Alert Notification Block 172
- Other Alert Block Properties 173
- Alert Dialog 177
- Alert Dialog Block 178
- Other Alert Dialog Block Properties 179

➢ Prompt 180
- Legacy 180
- Prompt Block 181
- Prompt Block Attributes 182

➢ Confirm 182
- Legacy 182

- Confirm Block 183
- Confirm Block Attributes 184
- Clipboard 185
  - Copy 185
  - Cut 190
  - Paste 196
- Fullscreen 198
  - Request 198
  - Exit 199
  - Is Fullscreen 199
    - Detect 200
    - Active 201
- Dates 202
  - Foundation 202
  - Date Default Methods 204
  - Date Get Methods 206
  - Date Get Gap Methods 210
  - UTC Get Methods 211
  - Date Set Methods 213
  - Locale Methods 216
- Redirect 218
  - Link (Default) 218
  - Replace 218
  - New Tab 219
  - Window 219
  - Download 221
  - Blob Download 222
  - IFrame 225
- Window 226
  - Foundation 226
  - Append 227
- JSON (HACK) 229
  - Introduction 229

- Parse (Copy) 229
  - Inline Method 229
  - Default Method 230
  - Block Method 231
- Stringify (Keep) 232
  - Default Method 232
  - Block Method 233
- WebStorage 234
- Session Storage 234
  - Set 234
  - Get 234
  - Remove 235
  - Clear 235
- Local Storage 237
  - Set 237
  - Get 237
  - Remove 238
  - Clear 238
- Debugging 240
- Window Console 240
  - Log (Default) 240
  - Clear 240
  - Assert 241
  - Count 241
  - Error 242
  - Warn 242
  - Info 243
  - Group 243
  - Group Collapsed 244
  - Group End 244
  - Time 245
  - Time End 245
  - Trace 245
  - Table 246

- ❖ Code View 247
  - ➢ Input Code (JaihoeScript) 247
  - ➢ Executable Code (JavaScript) 247
  - ➢ Library List (Jaihoe LibraryList) 248
  - ➢ Library Paths (Jaihoe LibraryPaths) 248
- ➢ Code Instants 249
  - ❖ Attribute Lists 249
  - ❖ Block Lists 250
- ➢ Custom 251
- ➢ Bad Practice 251

Libraries 252

- ➢ Convert 252
  - ❖ Foundation 252
  - ❖ Working Code 253
  - ❖ Converting List 253
  - ❖ Required Attribute Modifiers 254
  - ❖ Basic Attribute Modifiers 255
  - ❖ Rounding Attribute Modifiers 256
  - ❖ Exponential Attribute Modifiers 258
  - ❖ Forcing Attribute Modifiers 260
  - ❖ Conversion Variations 263
    - ➢ Volume conversion (convert volume) 263
    - ➢ Length conversion (convert length) 264
    - ➢ Weight and Mass conversion (convert weight) 265
    - ➢ Temperature conversion (convert temp) 266
    - ➢ Energy conversion (convert energy) 267
    - ➢ Area conversion (convert area) 268
    - ➢ Speed conversion (convert speed) 269
    - ➢ Time conversion (convert time) 270
    - ➢ Power conversion (convert power) 271
    - ➢ Pressure conversion (convert pressure) 272
    - ➢ Angle conversion (convert angle) 273
    - ➢ Basic Data conversion (convert data) 274

- Complex Data conversion (convert data) 275
- Currency 276
  - Foundation 276
  - Currency List 276
  - Currency List Attribute Modifiers 277
  - Preview (Currency Exchange) 277
  - Working Code (Currency Exchange) 278
  - Required Attribute Modifiers 278
  - Basic Attribute Modifiers 278
  - Rounding Attribute Modifiers 280
  - Currency Variations 282
- Fun 283
  - Foundation 283
  - Fun Text 283
    - Background Overlay Text 283
    - Blinking Text 284
    - Rainbow Text 285
  - Fun Title 286
    - Scrolling Title 286
    - Blinking Title 287
  - Fun Behavior 288
    - Smooth Go to Top 288
    - To Attract 289
    - To Repel 290

INTRODUCTION

Jaihoe is a manifesting programming language built on top of JavaScript which stands **for Jerwin's Attribute Interpreter in HTML Object Elements**. This project will depend mostly on the attributes, blocks and some tags of HTML, along with custom libraries that can make your page more **interactive**. Just include the Jaihoe main script and your good to go. Moreover, Jaihoe is meant to be **just for fun** and not for serious projects, as it uses **Script-Like Hypertext Markup Language** (**SLHTML**) syntax. Other JavaScript frameworks and libraries have the capability to modify the Document Object Model (**DOM**) but it requires additional skills to write it. With Jaihoe, coding a lot might be lengthy but as you go along, you can just copy a block snippet the code whereas the block of code is similar. For time efficiency in coding. Depending on what project your trying to accomplish.

Meanwhile for the history of Jaihoe as time passed by, while the developer was learning HTML and CSS, it did not take long as the developer also learned how to code basic JavaScript. After that, the developer was strangely inspired with PyScript which ships Python into the browser, as the developer started to gain ideas. Like how about if JavaScript was written in a different form of syntax and language. Thinking of what syntax and language it would be. The first idea was the syntax is similar to PHP but it ended up being similar to HTML. The development started on the 21st of August 2021 when the developer was scrolling on the internet and saw different kinds of HTML jokes. Furthermore, HTML was intentionally denoted as a programming language if you search on Google which is a joke for some people because it is a markup language. The developer said how about making this thing a reality perhaps because the developer sees the potential of what HTML can do.

Furthermore, the first development attempt was to fill the tag with attributes for simulation. The testing worked just fine until nested blocks was implemented as it only execute one block only not nested blocks. The second development was meant to be event-driven as it can execute code internally with nested block as well. Additionally, it can also be written externally by hosting it via localhost. Just beware of errors that might occur because some errors are not visible. Jaihoe is like turning JavaScript to Script-Like HTML which executes different functions. Now its your turn to create a casual code or project with it. Have fun using it and share it also with your friends for further testing.

Getting started with Jaihoe Blocks

- First of all, include the main Jaihoe script:

```
index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>Jaihoe Tutorial</title>
5
6      <!-- INCLUDE JAIHOE SCRIPT -->
7      <script type="application/javascript" src="jaihoe.js"></script>
8  </head>
<script type="application/javascript" src="jaihoe.js"></script>
```

  Take note that the SRC path might be different and there is nothing to worry about as long as the JavaScript library is imported and can be verified.

- To verify if the script is loaded, try to code the following:

```
10     <!-- JaihoeScript Block -->
11     <jh-script>
12         <j-about></j-about>
13     </jh-script>
14
15
16
17
18
19
20
21
```

```
<jh-script>
  <j-about></j-about>
</jh-script>
```

  Then try to test the JaihoeScript block above, in your browser.

- There are other implementations in showing the about section
  - Using the **element** approach
  - Using the **var** (variable) approach

    - By using the different approach you can target a specific element that you want to display in the about section. (Check on the next page)

- **About Section Implementations:**
  - The element approach.

```
<!-- JaihoeScript Block -->
<div id="about"></div>
<jh-script>
    <j-about element="#about" data-about-element></j-about>
</jh-script>
```

```
<div id="about"></div>
  <jh-script>
     <j-about element="#about" data-about-element></j-about>
  </jh-script>
```

  - The var (variable) approach.

```
<!-- JaihoeScript Block -->
<div id="about"></id>
<jh-script>
  <j-var data="varElement" data-eq data-var>
    <j-docget data-id="'about'" data-docget></j-docget>
  </j-var>
  <j-about var="varElement" data-about-var></j-about>
</jh-script>
```

```
<div id="about"></id>
  <jh-script>
     <j-var data="varElement" data-eq data-var>
       <j-docget data-id="'about'" data-docget></j-docget>
     </j-var>
     <j-about var="varElement" data-about-var></j-about>
  </jh-script>
```

Block Foundation

- There are two types of **block foundation**:
  - **Dependency Block**

```
<jh-dep>
   <!-- INSERT DEPENDENCY/LIBRARY BELOW -->

</jh-dep>
```

    - This block is where you write the dependencies and other libraries because some functions are unnecessary depending on the project you are working on.
    - You can replace **jh-dep** with j-dep, html-dep, or jaihoe-dep. However it is best to leave it as "**jh-dep**" because it will be the first to be executed.

  - **Script/Execution Block**

```
<jh-script>
    <!-- INSERT YOUR JAIHOE SCRIPT BELOW -->

</jh-script>
```

    - This block is where you write your JaihoeScript code to execute it on the browser.
    - You can replace **jh-script** with j-script, html-script, or jaihoe. However it is best to leave it as "**jh-script**" because it will be the first to be executed.

Block Structure

- For the common **block structure**
  - **Displaying Hello World in another element**

```
<j-hw element="#hwElement" data-hw-element></j-hw>
```

    - The "**j-hw**" is the tag directive, this is usually the open and closing tags.
    - The "**element**" attribute with value is the directive modifier, this usually modifies the values of the directive depending on its use. However, this is not used in closed-type block.
    - The "**data-hw-element**" attribute without value is the directive identifier. This is used if the block is used for different purposes but in some cases it is not needed.
    - The `<j-hw>` is the **opening tag directive** while the `</j-hw>` is the **closing tag directive**

Block Directive Distinction

- There are four types of script block distinction:
  - **Closed type Block**
    - This block does have attributes and it may have or not have blocks inside of it, even without attributes and type identifiers it will work as a function.
    - *Examples*:
      - Display about section
        ```
        <j-about></j-about>
        ```
      - Display Currency List
        ```
        <j-currency>
           <j-attr name="for" value="currency list" data-attr></j-attr>
           <j-attr name="target" value="#moneyList" data-attr></j-attr>
        </j-currency>
        ```
  - **Attribute-Only Block**
    - This block only contains one attribute to support mostly variables or other tunning values.
    - Example:
      - Set time value for a set timeout
        ```
        <j-timeout time="3000"></j-timeout>
        ```
        - Be careful to not separate the **closing** tag **directive** `</j-timeout>` in a new line because it **does not have** a directive identifier.
  - **Attribute-Type Block**
    - This is the **most common block** that has one attribute and one directive identifier to support a single instance of a block.
    - Example:
      - Declaring an inline variable
        ```
        <j-var data="online = false" data-var></j-var>
        ```
  - **Multiple Attribute-Type Block**
    - This block has multiple attribute and multiple type identifier to support multiple instances within the block
    - Example:
      - Calling a function **(Single Attribute to Many Type Identifiers)**
        ```
        <j-fx-call name="functAx" param data-fx-call></j-fx-call>
        ```
      - Attribute block for objects **(Many Attribute to Single Type Identifiers)**
        ```
        <j-attr name="for" value="fun behavior" data-attr></j-attr>
        ```

Block Patterns

- There are four types of **block patterns**:
  - **Non-Nested Block**
    - This block should not contain another block inside, this is also known as closed type block. (**reloading page**)

```
<j-reload></j-reload>
```

  - **Open-Nested Block**
    - This block may have or may not have another single or multiple block inside (**variable containing the value of PI**)

```
<!-- NESTED -->
<j-var data="pi = " data-var>
    <j-math type="PI" data-math></j-math>
</j-var>
<!-- NON-NESTED -->
  <j-var data="pi = 3.14" data-var></j-var>
```

      - Depending on how you declare a variable, based on your design pattern

  - **Closed-Nested Block**
    - This block may have blocks inside but some of the blocks are it is fixed (**reload page with time**)

```
<j-set-time>
    <j-fx name="reloadFunctAx" param data-fx>
       <j-reload></j-reload>
    </j-fx>
    <j-timeout time="2000"></j-timeout>
</j-set-time>
```

      - The only block that you can remove above is the "**j-reload**" and the others are fixed.

  - **Attribute-Nested Block**
    - This block may only have attributes (display currency list)

```
<j-currency>
   <j-attr name="for" value="currency list" data-attr></j-attr>
   <j-attr name="target" value="#moneyList" data-attr></j-attr>
</j-currency>
```

      - This is applicable on object functions and mostly for imported library functions

  - **Split-Nested Block**
    - This block is has a separator inside, in this case its <j-if-state/> then below or above the separator can have single or multiple block inside

```
<j-if-block>
  <j-value data="x == 5 && y == 5" data-value></j-value>
 <j-if-state/>
  <j-set data="x = 0" data-set></j-set>
  <j-set data="y = 0" data-set></j-set>
</j-if-block>
```

Importing Dependencies

- There are two types of **importing** a dependency/library:
  - **Default-Path Import Block**

```
<jh-dep>
    <j-imp name="jaihoe_convert"></j-imp>
</jh-dep>
```

    - This import block **depends** on the "**libraries**" folder where the other library scripts are located.
    - The "libraries" folder (containing the library scripts) and the HTML page should be located at the same path

  - **Custom-Path Import Block**

```
<jh-dep>
    <j-imp name="jaihoe_convert" src="../libraries"></j-imp>
</jh-dep>
```

    - This import block relies on the path of the library scripts even the "**libraries**" folder was renamed or not just make sure the path is valid
    - The example above is the libraries folder was not renamed but the HTML page was moved into another folder. That another folder was located in the same folder along with the libraries folder.

External Block Foundation

- You can externally define the dependency/library or JaihoeScript block like this:
  - **index.html (Main page)**

```
<p id="currencyList"></p>
<jh-dep src="dependency.html"></jh-dep>
<jh-script src="script.html"></jh-script>
```

  - **dependency.html (Dependency/Library file)**

```
<!-- IMPORT DEPENDENCY -->
<j-imp name="jaihoe_currency"></j-imp>
```

  - **script.html (JaihoeScript file)**

```
<j-currency>
   <j-attr name="for" value="currency list" data-attr></j-attr>
   <j-attr name="target" value="#currencyList" data-attr></j-attr>
</j-currency>
```

  - **Preview:**

Errors

- If **an error** occurs, mostly the code your trying to **display** is **not visible**.
- The error is usually displayed on the console log.
- **Examples**:
  - Dependency/Library and JaihoeScript Block was not hosted in the localhost or in a web-hosting service.

  - Invalid or Incomplete Syntax.

Basic Syntax

- Here are the basic syntax that you would need:
  - **Starters**
    - **Displaying Hello World (j-hw) [Easter Egg]**
      - This is how you display hello world

```
<!-- JaihoeScript Code -->
<jh-script>
    <j-hw></j-hw>
</jh-script>
```

```
<jh-script>
    <j-hw></j-hw>
</jh-script>
```

      - You could also target it with **element** method (same result)

```
<div id="hwElement"></id>
<jh-script>
   <j-hw element="#hwElement" data-hw-element></j-hw>
</jh-script>
```

      - You could also target it with **var** method (same result)

```
<div id="hwElement"></id>
<jh-script>
  <j-var data="varElement" data-eq data-var>
    <j-docget data-id="'hwElement'" data-docget></j-docget>
  </j-var>
  <j-hw var="varElement" data-hw-var></j-hw>
</jh-script>
```

    - **Display About Jaihoe (j-about)**
      - This is how you display the about section

```
<!-- JaihoeScript Code -->
<jh-script>
    <j-about></j-about>
</jh-script>
```

```
<jh-script>
    <j-about></j-about>
</jh-script>
```

- You could also target with **element** method (same result) [About Section]

```
<div id="aboutElement"></id>
  <jh-script>
    <j-about element="#aboutElement" data-about-element></j-about>
  </jh-script>
```

- You could also target with **var** method (same result) [About Section]

```
<div id="aboutElement"></id>
   <jh-script>
     <j-var data="varElement" data-eq data-var>
       <j-docget data-id="'aboutElement'" data-docget></j-docget>
     </j-var>
     <j-about var="varElement" data-about-var></j-about>
   </jh-script>
```

❖ **Reload Page (j-reload)**

➢ **Literally reloads the page**

```
<j-reload></j-reload>
```

```
<button id="reloadPage">Reload Page</button>
<jh-script>
  <j-add-evt element="#reloadPage" data-elem-evtype="click"
    data-add-evt>
     <j-fx name="reloadPageFx" param data-fx>
        <j-reload></j-reload>
     </j-fx>
  </j-add-evt>
</jh-script>
```

- **Variable**
  - **Using “var” variable (j-var)**
    - This how to declare an inline “var” variable and use it

```
<!-- JaihoeScript Code -->
<p>I am <span id="age"></span> years old!</p>
<jh-script>
    <j-var data="x = 23" data-var></j-var>
    <j-write element="'#age'" data-write-output="x" data-write></j-write>
</jh-script>
```

```
<p>I am <span id="age"></span> years old!</p>
    <jh-script>
        <j-var data="x = 23" data-var></j-var>
        <j-write element="'#age'"
          data-write-output="x" data-write></j-write>
    </jh-script>
```

    - This is how to declare a block “var” variable and use it

```
<!-- JaihoeScript Code -->
<p>The absolute value of PI is <span id="pi"></span>!</p>
<jh-script>
    <j-var data="x" data-eq data-var>
        <j-math type="PI" data-math></j-math>
    </j-var>
    <j-write element="'#pi'" data-write-output="x" data-write></j-write>
</jh-script>
```

```
<p>The absolute value of PI is <span id="pi"></span>!</p>
    <jh-script>
        <j-var data="x" data-eq data-var>
            <j-math type="PI" data-math></j-math>
        </j-var>
        <j-write element="'#pi'"
          data-write-output="x" data-write></j-write>
    </jh-script>
```

❖ Using “let” variable (j-let)

- The “let” variable are limited within the scope unlike “var”, in this example the “var” variable can pass through the block scope while the “var” cannot so the value is still 15.

```
<!-- JaihoeScript Code -->
<p style="margin:0;">The difference of var and let:</p>
<p style="margin:0;">var = <span id="varVal"></span></p>
<p style="margin:0;">let = <span id="letVal"></span></p>

<jh-script>
    <j-var data="xVar = 15" data-var></j-var>
    <j-let data="xLet = 15" data-let></j-let>

    <j-block>
        <j-var data="xVar = 10" data-var></j-var>
        <j-let data="xLet = 10" data-let></j-let>
    </j-block>

    <j-write element="'#varVal'"
        data-write-output="xVar" data-write></j-write>
    <j-write element="'#letVal'"
        data-write-output="xLet" data-write></j-write>
```

```
<p style="margin:0;">The difference of var and let:</p>
    <p style="margin:0;">var = <span id="varVal"></span></p>
    <p style="margin:0;">let = <span id="letVal"></span></p>
    <jh-script>
        <j-var data="xVar = 15" data-var></j-var>
        <j-let data="xLet = 15" data-let></j-let>

        <j-block>
            <j-var data="xVar = 10" data-var></j-var>
            <j-let data="xLet = 10" data-let></j-let>
        </j-block>
        <j-write element="'#varVal'"
            data-write-output="xVar" data-write></j-write>
        <j-write element="'#letVal'"
            data-write-output="xLet" data-write></j-write>
    </jh-script>
```

- You could also declare a block “let” variable

```
<p style="margin:0;">I'm <span id="age"></span> years old!</p>
<jh-script>
   <j-let data="ageLT" data-eq data-let>
      <j-math type="sqrt" param="625" data-math-method></j-math>
   </j-let>
   <j-write element="'#age'" data-write-output="ageLT"
      data-write></j-write>
</jh-script>
```

❖ Using “const” variable (j-const)

- The “const” constant variable can be confusing, because it makes a non-object reference variables like numbers or string to be constant while object reference variable such as array, maps and other sets can change their properties.
- This is how you declare an inline “const” variable

```
<!-- JaihoeScript Code -->
<p>My name is <span id="name"></span>!</p>
<jh-script>
    <j-const data="name = 'Jerwin'" data-const></j-const>
    <j-write element="'#name'" data-write-output="name"
        data-write></j-write>
</jh-script>
```

```
<p>My name is <span id="name"></span>!</p>
   <jh-script>
      <j-const data="nameCT = 'Jerwin'" data-const></j-const>
      <j-write element="'#name'" data-write-output="nameCT"
          data-write></j-write>
   </jh-script>
```

- This is how you declare a block “const” variable

```
<p>My name is <span id="name"></span>!</p>
   <jh-script>
      <j-const data="nameCT" data-eq data-const>
        <j-value-dq data="John"></j-value-dq>
      </j-const>
      <j-write element="'#name'" data-write-output="nameCT"
          data-write></j-write>
   </jh-script>
```

❖ Setting/changing variable (j-set)

- Assigning an empty variable using the inline method

```
<!-- JaihoeScript Code -->
<p>I've aged, I am now <span id="age"></span> years old!</p>
<jh-script>
    <j-var data="age" data-var></j-var>
    <j-set data="age = 24" data-set></j-set>
    <j-write element="'#age'" data-write-output="age"
        data-write></j-write>
</jh-script>
```

```
<p>I've aged, I am now <span id="age"></span> years old!</p>
    <jh-script>
        <j-var data="age" data-var></j-var>
        <j-set data="age = 24" data-set></j-set>
        <j-write element="'#age'" data-write-output="age"
            data-write></j-write>
    </jh-script>
```

- Assigning an empty variable using the block method

```
<!-- JaihoeScript Code -->
<p>I've aged, I am now <span id="age"></span> years old!</p>
<jh-script>
    <j-var data="age" data-var></j-var>
    <j-set data="age" data-eq data-set>
        <j-value data="25" data-value></j-value>
    </j-set>
    <j-write element="'#age'" data-write-output="age"
        data-write></j-write>
</jh-script>
```

```
<p>I've aged, I am now <span id="age"></span> years old!</p>
    <jh-script>
        <j-var data="age" data-var></j-var>
        <j-set data="age" data-eq data-set>
            <j-value data="25" data-value></j-value>
        </j-set>
        <j-write element="'#age'" data-write-output="age"
            data-write></j-write>
    </jh-script>
```

- **Set/change block variables (j-set-block)**
  - There are some variables that has already closed because it is defined as a block method, this method lets you **set a variable** without **closing** the variable to avoid complications
  - **Example**:
    - Here is the wrong way of defining a time interval

```
<j-let data="count" data-eq data-let>
  <j-set-interval data-fx="countNum" data-interval="1000"
    data-set-interval></j-set-interval>
</j-let>
```

    - Because in **JavaScript** it looks like this:

jaihoe.js

```
let count =
  setInterval(countNum,1000);
;
```

      - It has double closing tag (semi-colon) within the code

  - Now to avoid complications we have to use **set block** like this:

```
<j-let data="count" data-let></j-let>
<j-set-block data="count" data-eq data-set-block>
  <j-set-interval data-fx="countNum" data-interval="1000"
    data-set-interval></j-set-interval>
</j-set-block>
```

  - Then in **JavaScript** it will look like this:

jaihoe.js

```
 let count;
count =
 setInterval(countNum,1000);
```

    - It has now a single closing tag (semi-colon) within the code
    - This is how you define a block variable

- **Variable Value (j-value)**
  - This used mostly used for block variables and there are three types
    - **Default Value**

```
<j-value data="25"></j-value>
<!--OR-->
<j-value data="25" data-value></j-value>
```

      - Suggested values are mostly numbers, objects etc.
    - **Single Quoted Value**

```
<j-value data-sq='I know "this" and "that" yeah!'
   data-value-sq></j-value>
```

      - Suggested values are strings with double quotes
    - **Double Quoted Value**

```
<j-value data-dq="This is Jerwin's Dictionary"
   data-value-dq></j-value>
```

      - Suggested values are strings with single quotes
    - **Value-Area Value**

```
<j-valuearea hidden>
   Name: ${variable}
</j-valuearea>
```

      - Suggested values are strings with single or double quotes with embedded expression like (${variable})
      - This is best for multi-line string expressions, and this is a template literal.

- **If-else Statement**
  - **Using if (j-if)**
    - Example of using if statement

```
<j-if data="minCal < 3" data-if>
    <j-set data="awrRet = 'gold'" data-set></j-set>
</j-if>
```

  - **Using else if (j-else-if)**
    - Example of using else if statement

```
<j-else-if data="minCal < 8" data-else-if>
    <j-set data="awrRet = 'silver'" data-set></j-set>
</j-else-if>
```

  - **Using else (j-else)**
    - Example of using else statement

```
<j-else>
   <j-set data="awrRet = 'bronze'" data-set></j-set>
</j-else>
```

❖ **All together (If-else statement)**

```
<!-- JaihoeScript Code -->
 <p style="margin:0;">John run about
    <span id="minGx">5</span> minutes!</p>
 <p style="margin:0;">and won the
    <span id="awrDx"></span> medal!</p>

  <jh-script>
       <j-var data="minCal" data-eq data-var>
        <j-number>
            <j-docget data-id="'minGx'" data-prop="innerHTML"
                data-docget-extend></j-docget>
        </j-number>
       </j-var>
       <j-var data="awrRet" data-var></j-var>

       <j-if data="minCal < 3" data-if>
          <j-set data="awrRet = 'gold'" data-set></j-set>
       </j-if>
       <j-else-if data="minCal < 8" data-else-if>
          <j-set data="awrRet = 'silver'" data-set></j-set>
       </j-else-if>
       <j-else>
          <j-set data="awrRet = 'bronze'" data-set></j-set>
        </j-else>

       <j-write element="'#awrDx'" method="write"
         output="awrRet" data-write></j-write>

  </jh-script>
```

- **Using if block (j-if-block)**
  - Example of using if block statement
    (best use for **block** code **condition statement**)

```
<j-if-block>
  <j-value data="x < 3" data-value></j-value>
 <j-if-state/>
  <j-set data="xRet = x+' is less than 3'" data-set></j-set>
</j-if-block>
```

- **Using else if block (j-else-if-block)**
  - Example of using else if block statement

```
<j-else-if-block>
  <j-value data="x > 3" data-value></j-value>
 <j-else-if-state/>
  <j-set data="xRet = x+' is greater than 3'" data-set></j-set>
</j-else-if-block>
```

- **All together (If-else block statement)**

```
<p id="numTest" style="margin:0;display:inline;">
Greater or Less than <b>Random</b>:
</p>
<jh-script>
    <j-let data="x" data-eq data-let><j-random param="1,5"
      data-random></j-random></j-let>
    <j-let data="xRet = ''"data-let></j-let>

    <j-if-block><j-value data="x < 3" data-value></j-value>
    <j-if-state/>
    <j-set data="xRet = x+' is less than 3'" data-set></j-set>
    </j-if-block>
    <j-else-if-block><j-value data="x > 3" data-value></j-value>
    <j-else-if-state/>
    <j-set data="xRet = x+' is greater than 3'" data-set></j-set>
    </j-else-if-block>
    <j-else>
    <j-set data="xRet = x+' is equal 3'" data-set></j-set>
    </j-else>

    <j-append element="'#numTest'" method="append" output="xRet"
      data-append></j-append>
</jh-script>
```

- **Switch Statement**
  - **Foundation (j-switch)**

```
<j-switch value="variable here" data-switch>
</j-switch>
```

    - This is how you declare a switch statement, which is an alternative to if-else statement
  - **Case (j-case)**

```
<j-case data="value here" data-case />
```

    - This is how you insert a case in a switch statement, to check if the condition is true.
  - **Break (j-break)**

```
<j-break/>
```

    - This is how you insert a break in a switch statement, so it will not go pass through the other cases.
  - **Default (j-default)**

```
<j-default/>
```

    - This is how you insert a default in a switch statement, if the value is not found.
  - **All together (Switch Statement)**

```
<!-- JaihoeScript Code -->
<p><span id="thisDay"></span>!</p>
<jh-script>
  <j-var data="dayTx" data-var></j-var>
   <j-switch value="new Date().getDay()" data-switch>
      <j-case data="0" data-case />
         <j-set data="dayTx = 'Today is Sunday'"
                data-set></j-set>
         <j-break/>
      <j-case data="6" data-case />
         <j-set data="dayTx = 'Today is Saturday'"
                data-set></j-set>
         <j-break/>
      <j-default/>
         <j-set data="dayTx = 'Looking forward to weekends'"
                data-set></j-set>
    </j-switch>

    <j-write element="'#thisDay'" data-write-output="dayTx"
      data-write></j-write>
</jh-script>
```

    - That was tested on **Monday** so it will fall below to default

- **Loops**
  - **For Loop (j-for)**
    - This is an example of a for loop below

```
<j-for start="let i = 1" scope="i <= 10" action="i++" data-for>
</j-for>
```

    - This is a working example of for loop (Ascending) above

```
<!-- JaihoeScript Code -->
<p>Ascending Numbers: <span id="acNumbers"></span></p>
<jh-script>
  <j-for start="let i = 1" scope="i<=10" action="i++" data-for>
      <j-append element="'#acNumbers'" method="append"
         output="i" data-append></j-append>
      <j-if data="i < 10" data-if>
      <j-append element="'#acNumbers'" method="append"
         output="`,`" data-append></j-append>
      </j-if>
    </j-for>
</jh-script>
```

    - This is a working example of for loop (Descending) above

```
<p>Descending Numbers: <span id="acNumbers"></span></p>
<jh-script>
  <j-for start="let j = 10" scope="j>0" action="j--" data-for>
     <j-append element="'#acNumbers'" method="append"
        output="j" data-append></j-append>
     <j-if data="j > 1" data-if>
       <j-append element="'#acNumbers'" method="append"
          output="`,`" data-append></j-append>
     </j-if>
  </j-for>
</jh-script>
```

- **For-In Loop (j-for data-in)**
  - This is an example of a **for-in loop** below (**x** represents the index of the element of the array or object)

```
<j-for start="let x" data-in="array/object" data-for>
</j-for>
```

  - This is a working example of **a for-in loop** below (**odd numbers**)

```
<p>Odd Numbers: <span id="acNumbers"></span></p>
<jh-script>
  <j-var data="oddNum = [1,3,5,7,9]" data-var></j-var>
  <j-for start="let i" data-in="oddNum" data-for>
      <j-append element="'#acNumbers'" method="append"
      output="oddNum[i]" data-append></j-append>
      <j-if data="i < 4" data-if>
      <j-append element="'#acNumbers'" method="append"
         output="`,`" data-append></j-append>
      </j-if>
  </j-for>
</jh-script>
```

- **For-Of Loop (j-for data-of)**
  - This is an example of a **for-of loop** below (**y** represents the current object/array element variable that is iterated)

```
<j-for start="let y" data-of="array/object" data-for>
</j-for>
```

  - This is a working example of a **for-of loop** below (**even numbers**)

- This is a working code example of a **for-of loop** below (**even numbers**)

```
<p>Even Numbers: <span id="acNumbers"></span></p>
<jh-script>
  <j-var data="evenNum = [2,4,6,8,10]" data-var></j-var>

  <j-for start="let curEvenNum" data-of="evenNum" data-for>
    <j-append element="'#acNumbers'" method="append"
    output="curEvenNum" data-append></j-append>
      <j-if data="curEvenNum < 9" data-if>
        <j-append element="'#acNumbers'" method="append"
           output="','" data-append></j-append>
      </j-if>
  </j-for>
</jh-script>
```

❖ While Loop (j-while)

- This is an example of a while loop, it checks the condition before executing the code inside and continuously execute if it meets the condition on repeat.

```
<j-while scope="condition" data-while>
</j-while>
```

- This is a working example of while loop.

```
<p>Numbers Visible by (5) are: <span id="visNumbers"></span></p>
<jh-script>
  <j-let data="hopNum = 5" data-let></j-let>
    <j-while scope="hopNum < 55" data-while>
        <j-append element="'#visNumbers'" method="append"
           output="hopNum" data-append></j-append>
        <j-if data="hopNum < 50" data-if>
            <j-append element="'#visNumbers'" method="append"
               output="', '" data-append></j-append>
        </j-if>
        <j-set data="hopNum += 5" data-set></j-set>
    </j-while>

</jh-script>
```

- **Do While Loop (j-do)**
  - This is an example of a do while loop, after execution of the block of code, below if it meets the condition it will repeat continuously.

```
<j-do>
   <!--START CODE HERE-->

   <!-- END CODE HERE-->
   <j-while-do scope="condition" data-while-do />
</j-do>
```

  - This is a working example of do while loop

```
<!-- JaihoeScript Code -->
<p>Numbers Visible by (10) are:
   <span id="visNumbers"></span></p>
<jh-script>
  <j-let data="hopNum = 10" data-let></j-let>
    <j-do>
        <j-append element="'#visNumbers'" method="append"
           output="hopNum" data-append></j-append>
        <j-if data="hopNum < 100" data-if>
            <j-append element="'#visNumbers'" method="append"
               output="', '" data-append></j-append>
        </j-if>
        <j-set data="hopNum += 10" data-set></j-set>
      <j-while-do scope="hopNum < 110" data-while-do />
    </j-do>

</jh-script>
```

- **Continue and Break (j-continue …)**
  - You can put continue or break inside loops

```
<j-continue/>
<j-break/>
```

- **Functions**
  - **Making a function (j-fx)**
    - This is how you create a basic function

```
<j-fx name="functionAx" param data-fx>
   <!-- START CODE HERE -->

   <!-- END CODE HERE -->
</j-fx>
```

  - **Calling a basic function (j-fx-call)**

```
<j-fx-call name="functionAx" param data-fx-call></j-fx-call>
```

    - This is a working example of using a **basic function**

```
<p style="color:red;" id="hwElement"></p>
<jh-script>
    <j-fx name="helloWorldAx" param data-fx>
        <j-write element="'#hwElement'" method="write"
           output="'Hello World!'" data-write></j-write>
    </j-fx>
    <j-fx-call name="helloWorldAx" param data-fx-call>
    </j-fx-call>
</jh-script>
```

  - **Working Example**
    - This is a working example of using a **function with parameters**

```
<p style="color:blue;" id="hwElement"></p>
<jh-script>
  <j-fx name="introduceAx" param="name,age" data-fx>
      <j-write element="'#hwElement'" method="write"
         output="`I'm ${name}, ${age} years old!`"
         data-write></j-write>
    </j-fx>
  <j-fx-call name="introduceAx" param="'John', 25"
       data-fx-call></j-fx-call>
</jh-script>
```

- **Objects**
  - **Foundation (j-object)**
    - This is how you create an object variable

```
<j-var data="objName" data-eq data-var>
   <j-object>
      <!-- insert attributes here-->
   </j-object>
</j-var>
```

  - **Attributes (j-attr)**
    - This is how you add an **attribute** equals to (firstName: "Name",)

```
<j-attr name="firstName" value="Name" data-attr></j-attr>
```

      - Replace **data-attr** with **data-attr-close** or **data-close** if the attribute will be the last attribute, to remove the comma
    - This is how you add a **single quote attribute** equals to (lastName: 'Name',)

```
<j-attr name="lastName" data-value-sq="Name" data-attr-sq>
</j-attr>
```

      - Replace **data-attr-sq** with **data-attr-sq-close** if the last attribute will be the last attribute, to remove the comma
    - This is how you add an **any attribute** equals to (age: 25,)

```
<j-attr name="age" data-value-any="25" data-attr-any>
</j-attr>
```

      - Replace **data-attr-any** with **data-attr-any-close** if the last attribute will be the last attribute, to remove the comma
    - This is how you add a **block attribute** equals to (fullName: function(){ return theFullName; },)

```
<j-attr name="fullName" data-value-block>
  <j-fx param data-fx>
     <j-return value="`${this.firstName} ${this.lastName}`"
        data-return />
  </j-fx>
    <j-attr data-attr-block />
</j-attr>
```

      - Remove the `<j-attr data-attr-block />` **block** if the last attribute will be the last attribute, to remove the comma

❖ **All together (Objects)**

```
<!-- JaihoeScript Code -->
<p id="studentInfoBlock"></p>
<jh-script>
<j-var data="student" data-eq data-var>
    <j-object>
    <j-attr name="firstName" value="John" data-attr></j-attr>
    <j-attr name="lastName" data-value-sq="Doe" data-attr-sq>
    </j-attr>
    <j-attr name="id" data-value-any="2022556522" data-attr-any>
    </j-attr>
    <j-attr name="fullName" data-value-block>
        <j-fx param data-fx>
            <j-return value="`${this.firstName} ${this.lastName}`"
              data-return />
        </j-fx>
        <j-attr data-attr-block />
    </j-attr>
    <j-attr name="age" data-value-any="25" data-attr-any-close>
    </j-attr>
    </j-object>
</j-var>

  <j-write element="'#studentInfoBlock'" method="write"
     output="'Name: '+student.fullName()+', Age: '+student.age"
     data-write></j-write>

    <j-br element="'#studentInfoBlock'" data-br />
    <j-append element="'#studentInfoBlock'" method="append"
      output="'Student No.: '+student.id" data-append>
    </j-append>

</jh-script>
```

- **Stringify**
  - Converts complex array or objects [object Object] to string.

```
<j-stringify param="object/array" data-stringify></j-stringify>
```

```
<p style="word-wrap: break-word;" id="stringifyTest">
  <span style="color:red;">Object:</span></p>
<jh-script>
  <j-var data="student" data-eq data-var>
    <j-object>
      <j-attr name="name" value="John" data-attr></j-attr>
      <j-attr name="age" data-value-any="23" data-attr-any-close>
      </j-attr>
    </j-object>
  </j-var>
  <j-append element="'#stringifyTest'" data-append-block>
    <j-stringify param="student" data-stringify></j-stringify>
  </j-append>
</jh-script>
```

  - You could also try for **array**, not just objects:

```
<p style="word-wrap: break-word;" id="stringifyTest">
 <span style="color:red;">Array:</span></p>
<jh-script>
 <j-let data="odd" data-eq data-let>
 <j-array>
  <j-arel data="1" data-arel></j-arel>
  <j-arel data="3" data-arel></j-arel>
 </j-array></j-let>
 <j-append element="'#stringifyTest'" data-append-block>
  <j-stringify param="odd" data-stringify></j-stringify>
 </j-append>
</jh-script>
```

- **Return**
  - Literally returns a value from a function

```
<j-return value="'Hello World!'" data-return></j-return>
```

```
<div id="rExample"></div>
<jh-script>
  <j-fx name="greet" param data-fx>
    <j-return value="'Hello World!'" data-return></j-return>
  </j-fx>
  <j-write element="'#rExample'" data-write-block>
    <j-fx-vcall name="greet" param data-fx-vcall></j-fx-vcall>
  </j-write>
</jh-script>
```

  - You can replace the **return block** with:
    - **Set Approach**

```
<j-set data data-ret="'Hello World!'" data-set></j-set>
```

    - This is **equivalent** to: return 'Hello World'; (with semicolon)

    - **Value Approach**

```
<j-value data data-ret="'Hello World!'" data-value></j-value>
```

    - This is **equivalent** to: return 'Hello World' (without semicolon)

- **Output**
  - **Direct Write and Append**
    - This is how you write in a page

```
<j-write output="'String!'+variable" data-write></j-write>
```

    - This is how you append in a page

```
<j-append output="'String!'+variable" data-append></j-append>
```

  - **Element Write and Append**
    - This is how you write in a div using element method

```
<j-write element="'#element'" method="write"
   output="'String!'+variable" data-write></j-write>
```

    - This is how you append in a div using element method

```
<j-append element="'#element'" method="append"
output="'String!'+variable" data-append></j-append>
```

  - **Var Write and Append**
    - First define the variable **element target**

```
<j-var data="varElem" data-eq data-var>
   <j-docget data-id="'element'" data-docget></j-docget>
</j-var>
```

    - Then write the following below:
      - This is how you write in a div using var method

```
<j-write var="varElem" method="write"
  output="'String!'+variable" data-write></j-write>
```

      - This is how you append in a div using var method

```
<j-append var="varElem" method="append"
  output="'String!'+variable" data-append></j-append>
```

  - **Element Block Write/Append**
    - This is how you write in a div using element-block method

```
<j-write element="'#element'" data-write-block>
    <j-valuearea hidden>
        Hello ${variable}
    </j-valuearea>
</j-write>
```

      - Just replace the **element** to **var** with the element target
    - This is how you append in a div using block-element method

```
<j-append element="'#element'" data-append-block>
    <j-valuearea hidden>
        I'm ${age} years old
    </j-valuearea>
</j-append>
```

      - Just replace the **element** to **var** also with the element target

❖ Direct Block Write/Append

➢ If you want to display it directly just remove the **element** or **var**

```
<j-write data-write-block>
  <j-valuearea hidden>
      Hello ${variable}
  </j-valuearea>
</j-write>

<j-append data-append-block>
  <j-valuearea hidden>
      I'm ${age} years old
  </j-valuearea>
</j-append>
```

❖ Working Example (j-write & j-append)

```
<!-- JaihoeScript Code -->
<p id="waExample"></p>
<jh-script>
  <j-var data="name = 'John'" data-var></j-var>
  <j-var data="age = '25'" data-var></j-var>

   <j-write element="'#waExample'" method="write"
     output="`I'm ${name}`" data-write></j-write>
   <j-append element="'#waExample'" method="append"
     output="`, ${age} years old!`" data-append></j-append>
</jh-script>
```

➢ Sample code of using element **write** and **append**

❖ Omit Method Write and Append

- Instead of using **method write**

```
<j-write element="'#element'" method="write"
   output="'String!'+variable" data-write></j-write>
```

  - The **shorthand code** is:

```
<j-write element="'#element'"
  data-write-output="'String!'+variable" data-write></j-write>
```

- Instead of using **method append**

```
<j-append element="'#element'" method="append"
output="'String!'+variable" data-append></j-append>
```

  - The **shorthand code** is:

```
<j-append element="'#element'"
  data-append-output="'String!'+variable" data-append></j-append>
```

❖ Working (j-write & j-append) No Method

```
<p id="omwaExample"></p>
<jh-script>
   <j-write element="'#omwaExample'"
     data-write-output="'Hello '" data-write></j-write>
   <j-append element="'#omwaExample'"
     data-append-output="'World!'" data-append></j-append>
</jh-script>
```

❖ Var, Element, or Direct New Line (j-br)

➢ This is how you use a new line directly, just insert:

```
<j-br/>
```

```
<!-- JaihoeScript Code -->
<jh-script>
  <j-write output="'This is line one!'" data-write></j-write>
    <j-br/>
  <j-append output="'This is line two!'" data-append></j-append>
</jh-script>
```

➢ This is how you use a new line using element method

```
<j-br element="'#element" data-br />
```

```
<!-- JaihoeScript Code -->
<p style="margin:0;" id="bTest"></p>
<jh-script>
  <j-write element="'#bTest'" method="write"
    output="'Line one to go!'" data-write></j-write>
    <j-br element="'#bTest'" data-br />
  <j-append element="'#bTest'" method="append"
    output="'Line two to go!'" data-append></j-append>
</jh-script>
```

- This is how you use a new line using **var** method, after setting the variable element then use it like this

```
<j-br var="variableElement" data-br />
```

```
<!-- JaihoeScript Code -->
<p style="margin:0;" id="cDTest"></p>
<jh-script>

    <!-- Define the variable element first -->
    <j-var data="cTest" data-eq data-var>
        <j-docget data-id="'cDTest'" data-docget></j-docget>
     </j-var>

  <j-write var="cTest" method="write"
    output="'Line one to strike!'" data-write></j-write>
    <j-br var="cTest" data-br />
  <j-append var="cTest" method="append"
    output="'Line two to strike!'" data-append></j-append>

</jh-script>
```

- **Document Selector**
  - **Get Selectors (j-docget)**
    - These are the different types of document **get selectors**:
      - For getting element by id

        ```
        <j-docget data-id="'element'" data-docget></j-docget>
        ```

      - For getting element by tag name

        ```
        <j-docget data-tname="'element'" data-docget></j-docget>
        ```

      - For getting element by class name

        ```
        <j-docget data-cname="'element'" data-docget></j-docget>
        ```

      - For document query selector

        ```
        <j-docget data-qsel="'element'" data-docget></j-docget>
        ```

      - For document query selector all

        ```
        <j-docget data-qsela="'element'" data-docget></j-docget>
        ```

- **Self Selectors**
  - These are the different types of document **self selectors**:
    - For targeting the self document
      ```
      <j-docget data-self="title" data-docget-extend></j-docget>
      ```
    - For targeting the self document body
      ```
      <j-docget data-body="innerHTML" data-docget-extend></j-docget>
      ```
    - For targeting the self document head
      ```
      <j-docget data-head=" tagName" data-docget-extend></j-docget>
      ```

- **Property Extend (data-docget-extend)**
  - The property extend method is to get the property of the element depending on its type of element.

  - **Example #1**: This is an **input box** and its **element id** is **passwordHolder**, when the user enter something it will serve as a **property**, that property will be the **value** of the text box.
    - So getting the property value of the text box is shown below.
      ```
      <input id="passwordHolder" placeholder="Password"
        type="password" />
      ```


      ```
      <j-docget data-id="'passwordHolder'" data-prop="value"
        data-docget-extend></j-docget>
      ```
      - Enclose it with a **variable element** to get the input.

  - **Example #2**: This is a paragraph with an element id of **greetingHolder**, to get the text of the paragraph, use the **innerHTML** property
    ```
    <p id="greetingHolder">Nice to see you my friend.</p>
    ```


    ```
    <j-docget data-id="'greetingHolder'" data-prop="innerHTML"
      data-docget-extend></j-docget>
    ```
    - Enclose it with a **variable element** to get the text: "Nice to see you my friend"

- **Property Apply (apply="")**
  - The property apply method is to set the property of the targeted element depending on its type of element
  - **Example #1**: Changing the color of the text of the paragraph
    - To change it, the data property should be **style.color**, then the data equivalent could be **any color** you want in this case its **red** then indicate a **data-docget-apply** attribute at the end.

```
<j-docget data-id="'colorTest'" data-prop="style.color"
  data-eq="'red'" data-docget-apply></j-docget>
```

```
<p style="color:blue;" id="colorTest">This is color blue.</p>
<jh-script>

  <j-set-time>
    <j-fx param data-fx>
      <j-docget data-id="'colorTest'" data-prop="style.color"
        data-eq="'red'" data-docget-apply></j-docget>
      <j-write element="'#colorTest'" method="write"
        output="'This is now color red!'" data-write></j-write>
    </j-fx>
    <j-timeout time="2000"></j-timeout>
 </j-set-time>

</jh-script>
```

  - **Example #2**: Changing the DIV Background to rainbow (preview)

- **Example #2**: Changing the DIV Background to rainbow (code)

```
<!-- JaihoeScript Code -->
<div id="colorTest" style="color:white;font-size:40px;">
  Rainbow Background
</div>

<jh-script>
    <j-var data="rainbow" data-eq data-var>
      <j-valuearea hidden>
        linear-gradient(to right, red, orange, yellow,
        green, blue, indigo, violet)
      </j-valuearea>
    </j-var>
    <j-set-time>
      <j-fx param data-fx>
        <j-docget data-id="'colorTest'" data-prop="style.background"
        data-eq="rainbow" data-docget-apply></j-docget>

        <j-docget data-id="'colorTest'" data-prop="style.textAlign"
        data-eq="'center'" data-docget-apply></j-docget>
      </j-fx>
      <j-timeout time="2000"></j-timeout>
   </j-set-time>
</jh-script>
```

- **Property Apply Action (apply="action")**
  - The property apply action sets an action to the targeted element which includes a closed parameter methods depending on its use

```
<j-docget data-id="'contentInput'" data-prop="select"
  data-docget-apply="action"></j-docget>
```

```
<input id="contentInput" type="text" value="Select text from input">
<button id="selectText" type="button">Select</button><br/>
<jh-script>
  <j-add-evt element="#selectText" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
      <j-docget data-id="'contentInput'" data-prop="select"
        data-docget-apply="action"></j-docget>
    </j-fx>
  </j-add-evt>
</jh-script>
```

- **Property Apply Set (apply="set")**
  - The property apply set also makes an action to the targeted element but includes an open parameter depending on its use
  - S

```
<div id="spanCheck"><span id="mySPAN">This has span: </span></div>
<div id="noSpanCheck"></div>

<jh-script>
  <j-let data="divCont" data-let></j-let>
  <j-let data="span" data-eq data-let>
    <j-docget data-id="'mySPAN'" data-docget></j-docget>
  </j-let>
  <!-- CHANGE THE ID to spanCheck or noSpanCheck -->
  <j-set-block data="divCont" data-eq data-set-block>
    <j-docget data-id="'spanCheck'" data-prop="contains"
      param="span" data-docget-apply="set"></j-docget>
  </j-set-block>
  <j-append element="'#mySPAN'" method="append"
    output="divCont" data-append></j-append>
</jh-script>
```

- **Document Attributes**
  - **Set Attribute**
    - The method is to set the attribute of the targeted element, if the attribute does not exist, it will still be added

```
<j-docget data-id="'colorTest'" data-set-attr="'style'"
  data-attr-val="'color:blue;'" data-docget-apply="attr">
</j-docget>
```

```
<h1 id="colorTest">Hello World</h1>
<jh-script>
  <j-docget data-id="'colorTest'" data-set-attr="'style'"
    data-attr-val="'color:blue;'" data-docget-apply="attr">
  </j-docget>
</jh-script>
```

    - From the text color black, it will change to blue via set attribute

❖ Has Attribute

➢ The method is to check if the targeted element has a certain attribute of the entry, if an attribute is found it will return true, if not it will return false

```
<j-docget data-id="'idCheck'" data-has-attr="'title'"
  data-docget-apply="attr"></j-docget>
```

```
<p id="idCheck" title="Has title attribute?">If this has (title) attribute:
</p><p id="noIdCheck"></p>

<jh-script>
  <j-let data="hasAttr" data-let></j-let>
  <j-set-block data="hasAttr" data-eq data-set-block>
    <!-- You can replace idCheck with noIdCheck -->
    <j-docget data-id="'idCheck'" data-has-attr="'title'"
      data-docget-apply="attr"></j-docget>
  </j-set-block>
  <j-append element="'#idCheck'" method="append"
    output="hasAttr" data-append></j-append>
</jh-script>
```

❖ Get Attribute

➢ The method is to get the attribute value of the targeted element, if the attribute does not exist, it will return null

```
<j-docget data-id="'attrWR'" data-get-attr="'data-text'"
  data-docget-apply="attr"></j-docget>
```

```
<p id="attrWR" data-text="Hello World"><b>Get Attribute Text: </b></p>
<jh-script>
  <j-let data="textAttr" data-let></j-let>
  <j-set-block data="textAttr" data-eq data-set-block>
    <j-docget data-id="'attrWR'" data-get-attr="'data-text'"
      data-docget-apply="attr"></j-docget>
  </j-set-block>
  <j-append element="'#attrWR'" method="append"
    output="textAttr" data-append>
    </j-append>
</jh-script>
```

- **Event Listener**
  - **Element (j-add-event element)**
    - This is how you create an event listener using element method

```
<j-add-evt element="element" data-elem-evtype="event"
  data-add-evt>
   <j-fx param data-fx>
       <!-- INSERT CODE HERE -->
   </j-fx>
</j-add-evt>
```

  - **Var (j-add-event var)**
    - This is how you create an event listener using var method

```
<j-add-evt var="varEement" data-evtype="event"
  data-add-evt>
   <j-fx param data-fx>
       <!-- INSERT CODE HERE -->
   </j-fx>
</j-add-evt>
```

- **Event Single Function Call (j-fx-evt)**
  - Instead of using inline function inside an event listener, you can use the **j-fx-event** method.

```
<j-add-evt element="element" data-elem-evtype="click"
    data-add-evt>
   <j-fx-evt name="functionName" data-fx-evt></j-fx-evt>
  </j-add-evt>
```

  - Then declare the function event

```
<j-fx name="functionName" param data-fx>
   <!-- INSERT CODE HERE -->
</j-fx>
```

- **All together (Event Listener)**
  - This is an example of making the text to **bold** if **clicked**

- **All together (Event Listener)**
  - Here is the working code of making the text to **bold** if **clicked**

```
<p id="boldFx">Click to make the font bold!</p>
<jh-script>
  <j-add-evt element="#boldFx" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
      <j-docget data-id="'boldFx'" data-prop="style.fontWeight"
        data-eq="'bold'" data-docget-apply></j-docget>
     </j-fx>
  </j-add-evt>
</jh-script>
```

  - You can also do it in a single function call (no parameters)

```
<p id="boldFx">Click to make the font bold!</p>
<jh-script>
  <j-add-evt element="#boldFx" data-elem-evtype="click"
    data-add-evt>
    <j-fx-evt name="boldFx" data-fx-evt></j-fx-evt>
  </j-add-evt>

 <j-fx name="boldFx" param data-fx>
   <j-docget data-id="'boldFx'" data-prop="style.fontWeight"
   data-eq="'bold'" data-docget-apply></j-docget>
 </j-fx>
</jh-script>
```

- **Math**
  - **Properties**
    - **To get the approximate value of Square Root of 2**

```
<j-math type="SQRT2" data-math></j-math>
```

```
<!-- JaihoeScript Code -->
<p id="mathTest"></p>
<jh-script>
  <j-write element="'#mathTest'" method="write"
    output="'Square root of 2 is approximately equal to: '"
    data-write></j-write>
  <j-br element="'#mathTest'" data-br />
  <j-append element="'#mathTest'" data-append-block>
    <j-math type="SQRT2" data-math></j-math>
  </j-append>
</jh-script>
```

    - You can change the **type** with E, PI, SQRT1_2, LN2, LN10, LOG2E, LOG10E aside from SQRT2

  - **Math Methods**
    - **Single Value Methods**
      - These are math methods that accept only one value

```
<j-math type="sqrt" param="64" data-math-method></j-math>
```

      - This is the preview of getting the square root of 64

- **Single Value Methods**
  - This is the code of getting the square root of 64

```
<!-- JaihoeScript Code -->
<p id="mathTest"></p>
<jh-script>
  <j-write element="'#mathTest'" method="write"
    output="'The square root of 64 is '" data-write></j-write>
  <j-append element="'#mathTest'" data-append-block>
    <j-math type="sqrt" param="64" data-math-method></j-math>
  </j-append>
</jh-script>
```

    - You can change the **type** with abs, acos, acosh, asin, asinh, atan, atanh, cbrt, ceil, clz32, cos, cosh, exp, expm1, floor, fround, log, log10, log1p, log2, round, sign, sin, sinh, **sqrt**, tan, tanh and trunc

- **Double Value Methods**
  - These are math methods that accept two values

```
<j-math type="pow" param="4,3" data-math-method></j-math>
```

```
<!-- JaihoeScript Code -->
<p id="mathTest">4<sup>3</sup> = 4 x 4 x 4 = </p>

<jh-script>
  <j-append element="'#mathTest'" data-append-block>
    <j-math type="pow" param="4,3" data-math-method></j-math>
  </j-append>
</jh-script>
```

    - You can change the type with atan2 or **pow**

- **Multi Value Methods**
  - These are math methods that accept multiple values

    - This is the preview of getting the highest number in this set

- **Multi Value Methods**
  - This the code of getting the highest number in the set (5,3,2,6,4,1)

```
<!-- JaihoeScript Code -->
<p id="mathTest">The highest number in (5,3,2,6,4,1) is </p>
<jh-script>

  <j-append element="'#mathTest'" data-append-block>
    <j-math type="max" param="5,3,2,6,4,1"
      data-math-method></j-math>
  </j-append>
</jh-script>
```

- **Random Number**
  - **Random Entry Value**
    - If you just want to generate random number

```
<p id="mathTest">The Random Number is </p>
<jh-script>
   <j-append element="'#mathTest'" data-append-block>
     <j-random data-random></j-random>
   </j-append>
</jh-script>
```

    - You can also change the `<j-random></j-random>` with

```
<j-random param data-random></j-random>
<j-random param="" data-random></j-random>
<j-random param=" " data-random></j-random>
<j-random param="a" data-random></j-random>
```

  - **Single Entry Value**
    - If you just want to generate a random number between 0 to your specified number.
    - For example: Random number from 0 to 5, the entry is "5" (preview)

- **Single Entry value**
  - For example: Random number is from 0 to 5, the entry is “5” (**code**)

```
<p id="mathTest">The Random Number is </p>
<jh-script>
  <j-append element="'#mathTest'" data-append-block>
    <j-random param="5" data-random></j-random>
  </j-append>
</jh-script>
```

  - Just change the **param** attribute to 5, or any numeric value you want

- **Double Entry Value**
  - If you want to generate number between your specified ranged number
  - For example: Random number is from **2 to 8**, the entry is “**2,8**”

  - If random number is from **1 to 10**, the entry is “**1,10**” or “**10,1**”

- **Random Sentence**
  - Generates a random sentence depending on its variant
    - **Sentence Generator**
      - Constructs a sentence which depends on its size inputted

      - The results may differ because its random

- **Random Sentence**
  - **Sentence Generator**
    - Here is the working code of using the sentence generator:

```
<div id="sHolder" style="margin-bottom:30px;text-align:justify;text-justify:inter-word;"></div>

<jh-script>
  <j-write element="'#sHolder'" data-write-block>
  <j-random type="sentence" param="2" data-random></j-random>
  </j-write>
</jh-script>
```

  - **Lorem Generation**
    - Constructs a lorem ipsum sentence which depending on its size inputted

```
<div id="lHolder" style="margin-bottom:30px;text-align:justify;text-justify:inter-word;"></div>

<jh-script>
  <j-write element="'#lHolder'" data-write-block>
  <j-random type="lorem" param="2" data-random></j-random>
  </j-write>
</jh-script>
```

- **String**
  - **Length**
    - To get the length of the string (snippet code)

```
<j-string data="'string'" data-length></j-string>
```

- **Length**
  - To get the length of the string, here is the working code

```
<p id="strTest">(A to Z).length is </p>
<jh-script>
  <j-let data="str = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'" data-let>
  </j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" data-length></j-string>
  </j-append>
</jh-script>
```

- **Default Methods**
  - **Slice**

    Extracts a part of a string to a new extracted part of the string

```
<j-string data="str" method="string" type="slice" param="7,13"
data-string-method></j-string>
```

```
<!-- JaihoeScript Code -->
<p id="strTest">(Apple, Banana, Kiwi).slice is </p>
<jh-script>
  <j-let data="str = 'Apple, Banana, Kiwi'" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" method="string" type="slice"
      param="7,13" data-string-method></j-string>
  </j-append>
</jh-script>
```

  - **Substring**
    - Similar to slice but the difference is that start and end values as negative returns as 0.
    - Here is the working code:

```
<!-- JaihoeScript Code -->
<p id="strTest">(Apple, Banana, Kiwi).substring is </p>
<jh-script>
  <j-let data="str = 'Apple, Banana, Kiwi'" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" method="string" type="substring"
      param="7,13" data-string-method></j-string>
  </j-append>
</jh-script>
```

- **Substring**
  - Here is the preview of the working code:

```
<j-string data="str" method="string" type="substring"
  param="7,13" data-string-method></j-string>
```

- **Substr**
  - This is also similar to slice but the difference is the extracted part of the second parameter specifies the length

```
<j-string data="str" method="string" type="substr" param="7,6"
data-string-method></j-string>
```

```
<p id="strTest">(Apple, Banana, Kiwi).substr is </p>
<jh-script>
  <j-let data="str = 'Apple, Banana, Kiwi'" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" method="string" type="substr"
      param="7,6" data-string-method></j-string>
  </j-append>
</jh-script>
```

  - **Replace**
    - Replaces a part a string **(once)** within the specified word or values
    - Here is the working code:

```
<p id="strTest"></p>
<jh-script>
  <j-let data="str = 'Please try React!'" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" method="string" type="replace"
      param="'React','Jaihoe'" data-string-method></j-string>
  </j-append>
</jh-script>
```

- **Replace**
  - Here is the preview:

```
<j-string data="str" method="string" type="replace"
  param="'React','Jaihoe'" data-string-method></j-string>
```

  - **ReplaceAll**
    - Replaces within the scope of string **(multiple times)** within the specified word or values

```
<j-string data="str" method="string" type="replaceAll"
  param="'Dog','Cat'" data-string-method></j-string>
```

```
<p id="strTest"></p>
<jh-script>
  <j-let data="str = 'Please feed the Dog! Dog can get hungry too.'"
    data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" method="string" type="replaceAll"
      param="'Dog','Cat'" data-string-method></j-string>
  </j-append>
</jh-script>
```

  - **Uppercase**
    - Converts the string to uppercase, here is the preview

```
<j-string data="str" data-string-uppercase></j-string>
```

- **Uppercase**
  - Here is the working code

```
<!-- JaihoeScript Code -->
<p id="strTest"></p>
<jh-script>
  <j-let data="str = 'this will turn to uppercase.'"
    data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" data-string-uppercase></j-string>
  </j-append>
</jh-script>
```

- **Lowercase**
  - Converts the string to lowercase, literally opposite to uppercase

```
<j-string data="str" data-string-lowercase></j-string>
```

```
<!-- JaihoeScript Code -->
<p id="strTest"></p>
<jh-script>
  <j-let data="str = 'THIS WILL TURN TO LOWERCASE.'"
    data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" data-string-lowercase></j-string>
  </j-append>
</jh-script>
```

- **Concat**
  - Strings are joined depending on its delimiter

```
<j-string data="text1" method="string" type="concat"
  param="' ', text2" data-string-method></j-string>
```

- **Concat**
  - Here is the working code of the preview

```
<!-- JaihoeScript Code -->
<p id="strTest"></p>
<jh-script>
  <j-let data="text1 = 'Hello'" data-let></j-let>
  <j-let data="text2 = 'World'" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="text1" method="string" type="concat"
      param="' ', text2" data-string-method></j-string>
  </j-append>
</jh-script>
```

- **Trim**
  - Whitespaces in the string from both sides are removed

```
<j-string data="str" data-string-trim></j-string>
```

```
<!-- JaihoeScript Code -->
<p id="strTest"></p>
<jh-script>
  <j-let data="str = '    Hello World!    '" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" data-string-trim></j-string>
  </j-append>
</jh-script>
```

You can replace `data-string-trim` with `data-string-trim="both"`

- **Trim Start**
  - Whitespaces from the start of the string are removed

```
<j-string data="str" data-string-trim="start"></j-string>
```

- **Trim Start**
  - Here is the working code of the preview

```
<p id="strTest"></p>
<jh-script>
  <j-let data="str = '     Start Engine!'" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" data-string-trim="start"></j-string>
  </j-append>
</jh-script>
```

- **Trim End**
  - Whitespaces in the string from the end of the string are removed

```
<j-string data="strA" data-string-trim="end"></j-string>
```

```
<!-- JaihoeScript Code -->
<p id="strTest"></p>
<jh-script>
  <j-let data="strA = 'End Engine!       '" data-let></j-let>
  <j-let data="str1" data-eq data-let>
    <j-string data="strA" data-length></j-string>
  </j-let>

  <j-let data="strB" data-eq data-let>
    <j-string data="strA" data-string-trim="end"></j-string>
  </j-let>
  <j-let data="str2" data-eq data-let>
    <j-string data="strB" data-length></j-string>
  </j-let>

  <j-append element="'#strTest'" data-append-block>
    <j-string data="strB+' ('+str1+', '+str2+')'"
      data-string></j-string>
  </j-append>

</jh-script>
```

- **PadStart**
  - Adds a padding from the start of the string with a specific character

```
<j-string data="str" method="string" type="padStart"
  param="4,'x'" data-string-method></j-string>
```

```
<p id="strTest"></p>
<jh-script>
  <j-let data="str = '5'" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" method="string" type="padStart"
      param="4,'x'" data-string-method></j-string>
  </j-append>
</jh-script>
```

- **PadEnd**
  - Adds a padding from the end of the string with a specific character

```
<j-string data="str" method="string" type="padEnd"
        param="4,'x'" data-string-method></j-string>
```

```
<p id="strTest"></p>
<jh-script>
  <j-let data="str = '5'" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" method="string" type="padEnd"
      param="4,'x'" data-string-method></j-string>
  </j-append>
</jh-script>
```

- **CharAt**
  - Gets the character from the part of the string on the specified index

```
<j-string data="str" method="string" type="charAt"
  param="0" data-string-method></j-string>
```

```
<p style="margin:0;">
  <b>CharAt(<span style="color:red;">0</span>)</b>
  in <b>HELLO WORLD</b> is
  <span style="color:blue" id="strTest"></span>
</p>
<jh-script>
  <j-let data="str = 'HELLO WORLD'" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" method="string" type="charAt"
      param="0" data-string-method></j-string>
  </j-append>
</jh-script>
```

- **CharCodeAt**
  - Gets the character code from the part of the string on the specified index

```
<j-string data="str" method="string" type="charCodeAt"
  param="0" data-string-method></j-string>
```

```
<p style="margin:0;">
  <b>CharCodeAt(<span style="color:red;">0</span>)</b>
  in <b>HELLO WORLD</b> is
  <span style="color:blue" id="strTest"></span>
</p>
<jh-script>
  <j-let data="str = 'HELLO WORLD'" data-let></j-let>
  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" method="string" type="charCodeAt"
      param="0" data-string-method></j-string>
  </j-append>
</jh-script>
```

- **Split**
  - Strings are split into characters depending on its delimiter and to be converted to array when method is applied

```
<j-string data="str" method="string"
  type="split" param="'.'" data-string-method></j-string>
```

```
<p id="strTest"></p>
<jh-script>
  <j-let data="str = 'H.e.l.l.o'" data-let></j-let>
  <j-const data="myArr" data-eq data-const>
    <j-string data="str" method="string"
      type="split" param="'.'" data-string-method></j-string>
  </j-const>

  <j-set data="str = ''" data-set></j-set>
  <j-for start="let x = 0" scope="x < myArr.length" action="x++"
    data-for>
    <j-set data="str += myArr[x]" data-add data-br data-set>
    </j-set>
  </j-for>

  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" data-string></j-string>
  </j-append>
</jh-script>
```

- **Search Methods**
  - **IndexOf**

    The index of the position of the first occurrence of the string will return depending on the query

```
<j-string data="str" method="string" type="indexOf"
param="'locate'" data-string-method></j-string>
```

  - **Last IndexOf**

    The index of the last occurrence of the string will return depending on the query

```
<j-string data="str" method="string" type="lastIndexOf"
param="'locate',15" data-string-method></j-string>
```

  - **Search**

    The string for a string (or a regular expression) it will be searched and return the position of the match

```
<j-string data="str" method="string" type="search"
param="'locate'" data-string-method></j-string>
```

  - **Match**

    The matching string (or a regular expression) will returned as an array. Take note that without global modifier, it will return the first match.

```
<j-string data="str" method="string" type="match" param="'ain'"
  data-string-method></j-string>
```

    - **Here is the preview of the match from:**
      - "The rain in SPAIN stays mainly plain"

- **Match**
  - Here is the working code of the preview:

```
<p id="strTest"><b>MATCH:</b> </p>
<jh-script>
  <j-let data="str = 'The rain in SPAIN stays mainly plain.'"
    data-let></j-let>
  <j-const data="myArr" data-eq data-const>
    <j-string data="str" method="string"
      type="match" param="'ain'" data-string-method></j-string>
  </j-const>

  <j-set data="str = ''" data-set></j-set>
  <j-for start="let x = 0" scope="x < myArr.length" action="x++"
    data-for>
    <j-set data="str += myArr[x]" data-set>
    </j-set>
  </j-for>

  <j-append element="'#strTest'" data-append-block>
    <j-value data="str + ' ('+myArr.length+')'"></j-value>
  </j-append>
</jh-script>
```

- **Match All**
  - All the matching string (or a regular expression) will returned as an array depending on the part of the case of the string.

```
<j-string data="str" method="string"
        type="matchAll" param='"Cats"' data-string-method></j-
string>
```

```
<p id="strTest"><b>MATCH:</b> </p>
<jh-script>
  <j-let data="str = 'I love cats. Cats are love. Cats are cool.'"
    data-let></j-let>
  <j-const data="myArr" data-eq data-const>
    <j-string data="str" method="string" type="matchAll"
      param='"Cats"' data-string-method></j-string></j-const>
    <j-append element="'#strTest'" data-append-block>
      <j-value data="Array.from(myArr)"></j-value>
    </j-append>
  </jh-script>
```

- **Includes**

```
<j-string data="str" method="string" type="includes"
param="'world'" data-string-method></j-string>
```

```
<p id="strTest"><b> IF </b>
   (Hello world, welcome to the universe)
   <b> INCLUDES </b>(world): </p>

  <jh-script>
    <j-let data="str = 'Hello world, welcome to the universe.'"
      data-let></j-let>

  <j-append element="'#strTest'" data-append-block>
    <j-string data="str" method="string" type="includes"
      param="'world'" data-string-method></j-string>
  </j-append>
</jh-script>
```

- **startsWith**
  - If the string starts with the defined value it will return as true

```
<j-string data="str" method="string" type="startsWith"
param="'Hello'" data-string-method></j-string>
```

  - **Here is the preview of the starts with "Hello" from:**
    - "Hello world, welcome to the universe"

- **startsWith**
  - Here is the working code of the preview:

```
<p id="strTest"><b>IF </b>
      (Hello world, welcome to the universe)
      <b> STARTSWITH </b>(Hello):
</p>
  <jh-script>
    <j-let data="str = 'Hello world, welcome to the universe.'"
      data-let></j-let>

      <j-append element="'#strTest'" data-append-block>
        <j-string data="str" method="string" type="startsWith"
          param="'Hello'" data-string-method></j-string>
      </j-append>
    </jh-script>
```

- **endsWith**
  - If the strings end with the defined value it will return as true

```
<j-string data="str" method="string" type="endsWith"
  param="'world'" data-string-method></j-string>
```

```
  <p id="strTest">
   <b>IF </b>(Hello world)<b> ENDSWITH </b>(world):
  </p>
  <jh-script>
    <j-let data="str = 'Hello world'" data-let></j-let>
    <j-append element="'#strTest'" data-append-block>
      <j-string data="str" method="string" type="endsWith"
        param="'world'" data-string-method></j-string>
    </j-append>
  </jh-script>
```

- **Extended Methods**
  - These methods are exclusive if Jaihoe is included.
    - **Case Type**
      - Returns the current case of the string

```
<j-string method="jaihoe" type="caseType" param="'helloword'"
data-string-method></j-string>
```

```
<p style="margin:0;" id="strTest">
 <span style="color:red;">Cases:</span>
</p>
<jh-script>
     <j-append element="'#strTest'" data-append-block>
      <j-value-rbr/>
      <j-string method="jaihoe" type="caseType"
        param="'helloword'" data-string-method></j-string>
      <j-value-br/>
      <j-string method="jaihoe" type="caseType"
        param="'HELLOWORLD'" data-string-method></j-string>
      <j-value-br/>
      <j-string method="jaihoe" type="caseType"
        param="'Hello World'" data-string-method>
      </j-string><j-value-br/>
       <j-string method="jaihoe" type="caseType"
         param="'Hello world'" data-string-method>
       </j-string><j-value-br/>
       <j-string method="jaihoe" type="caseType"
         param="'helloWorld'" data-string-method>
       </j-string><j-value-br/>
       <j-string method="jaihoe" type="caseType"
         param="'HelloWorld'" data-string-method>
       </j-string><j-value-br/>
       <j-string method="jaihoe" type="caseType"
         param="'heLlOWoRlD'" data-string-method></j-string>
     </j-append>
</jh-script>
```

- **Has Number**
  - Check if string has a number, the number of occurrences can be defined

```
<j-string method="jaihoe" type="hasNumber"
  param="'x123'" data-string-method></j-string>
```

```
<p style="margin:0;" id="strTest">
  <span style="color:red;">Has Number:</span>
</p>
<jh-script>
  <j-append element="'#strTest'" data-append-block>
   <j-value-rbr/>
   <j-value data="'If (x123) has 1 number: '" data-value>
     </j-value><j-value-add/>
   <j-string method="jaihoe" type="hasNumber"
     param="'x123'" data-string-method></j-string><j-value-br/>

   <j-value data="'If (x123) has 4 numbers: '" data-value>
     </j-value><j-value-add/>
     <j-string method="jaihoe" type="hasNumber"
       param="'x123',4" data-string-method></j-string>

  </j-append>
</jh-script>
```

- **Has Letters**
  - Check if string has a letter, the letter of occurrences can be defined

```
<j-string method="jaihoe" type="hasLetter"
  param="'abc123'" data-string-method></j-string>
```

- **Has Letters**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="strTest">
    <span style="color:red;">Has Letters:</span>
</p>
<jh-script>
<j-append element="'#strTest'" data-append-block>
 <j-value-rbr/>
 <j-value data="'If (abc123) has 1 letter: '" data-value>
   </j-value><j-value-add/>
 <j-string method="jaihoe" type="hasLetter"
   param="'abc123'" data-string-method></j-string><j-value-br/>

 <j-value data="'If (abc123) has 5 letters: '" data-value>
   </j-value><j-value-add/>
   <j-string method="jaihoe" type="hasLetter"
     param="'abc123',5" data-string-method></j-string>

</j-append>
</jh-script>
```

- **Has Symbols**
  - Check if string has a symbol, the symbol of occurrences can be defined

```
<j-string method="jaihoe" type="hasSymbol"
  param="'#Hello%'" data-string-method></j-string>
```

  - **For example**:
    - (#Hello%) has:
      - 2 symbols
      - 5 letters

  - **For checking:**
    - Check if string has 1 symbol
    - Check if string has 3 symbols

- **Has Symbols**
  - Check if string has a symbol, the symbol of occurrences can be defined

```
<p style="margin:0;" id="strTest">
    <span style="color:red;">Has Letters:</span>
</p>
<jh-script>
<j-append element="'#strTest'" data-append-block>
 <j-value-rbr/>
 <j-value data="'If (#Hello%) has 1 symbol: '" data-value>
   </j-value><j-value-add/>
 <j-string method="jaihoe" type="hasSymbol"
   param="'#Hello%'" data-string-method></j-string><j-value-br/>

 <j-value data="'If (#Hello%) has 3 symbols: '" data-value>
   </j-value><j-value-add/>
   <j-string method="jaihoe" type="hasSymbol"
     param="'#Hello%',3" data-string-method></j-string>

</j-append>
</jh-script>
```

- **Has Space**
  - Literally check if string has whitespaces

```
<j-string method="jaihoe" type="hasSpace"
   param="'Hello World'" data-string-method></j-string>
```

```
<p style="margin:0;" id="strTest">
  <span>If </span><b>(Hello World)</b>
  <b style="color:red;">has space</b><b>: </b>
</p>
<jh-script>
 <j-append element="'#strTest'" data-append-block>
   <j-string method="jaihoe" type="hasSpace"
     param="'Hello World'" data-string-method></j-string>
 </j-append>
</jh-script>
```

- **Reverse**
  - Literally reverse the string, if the string was already in reverse it will un-reverse it to the original state

```
<p style="margin:0;" id="strTest">
  <span>The reverse way of </span><b>(dlroW olleH)</b>
  <b style="color:red;">is</b><b>: </b>
</p>
<jh-script>
  <j-append element="'#strTest'" data-append-block>
    <j-string method="jaihoe" type="reverse"
      param="'dlroW olleH'" data-string-method></j-string>
  </j-append>
</jh-script>
```

- **Hex Conversion**
  - Converts string to hexadecimal and hexadecimal to string vice versa

```
<j-string method="jaihoe" type="strToHex" param="str"
  data-string-method></j-string>
<j-string method="jaihoe" type="hexToStr" param="hex"
  data-string-method></j-string>
```

- **Hex Conversion**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="strHexTest"><b>STR & HEX TEST:</b></p>
<jh-script>
    <j-let data="str = `John's Bakery`" data-let></j-let>
    <j-let data="hex" data-eq data-let>
    <j-string method="jaihoe" type="strToHex"
      param="str" data-string-method></j-string>
    </j-let>
    <j-let data="converted = " data-let>
    <j-string method="jaihoe" type="hexToStr"
      param="hex" data-string-method></j-string>
  </j-let>

  <j-append element="'#strHexTest'" data-append-block>
    <j-value-rbr/>
    <j-value data="'<b>String: </b>'+str" data-value></j-value>
     <j-value-br/>
    <j-value data="'<b>Hex: </b>'+hex" data-value></j-value>
     <j-value-br/>
    <j-value data="'<b>Converted: </b>'+converted" data-value>
     </j-value>
  </j-append>
</jh-script>
```

  - **What happened?**:
    - Each character in a string has a char code to be transform
    - The collection of char code is combined and form as a hexadecimal code
    - Reading the hexadecimal code is kind of the reverse where the char code is translated to a readable characters
    - Each readable characters is combined to make the original string to its original form

  - **Tips and Tricks:**
    - Hex conversion is best used for passing variables with quotes to parameters
    - Not in parameters but also in JSON which is also have quotes if are strings are used.

- **Unique Number**
  - Convert string to a unique number, you can also change the unique number to positive or negative

```
<j-string method="jaihoe" type="strToUID"
  param="'Hello world'" data-string-method></j-string>
<j-string method="jaihoe" type="strToUID"
  param="'Hello','-'" data-string-method></j-string>
<j-string method="jaihoe" type="strToUID"
  param="'Hello World!','+'" data-string-method></j-string>
```

```
<p style="margin:0;" id="strHexTest"><b>STR & UID:</b></p>
<jh-script>
  <j-let data="str = 'Hello World!'" data-let></j-let>
  <j-let data="uidN" data-eq data-let>
    <j-string method="jaihoe" type="strToUID"
      param="str" data-string-method></j-string>
  </j-let>
  <j-let data="uidP" data-eq data-let>
    <j-string method="jaihoe" type="strToUID"
      param="str,'+'" data-string-method></j-string>
  </j-let>

 <j-append element="'#strHexTest'" data-append-block>
  <j-value-rbr/>
  <j-value data="'<b>String: </b>'+str" data-value></j-value>
    <j-value-br/>
 <j-value data="'<b>UID (+): </b>'+uidP" data-value></j-value>
   <j-value-br/>
 <j-value data="'<b>UID (-): </b>'+uidN" data-value></j-value>
   </j-append>
</jh-script>
```

- **For example**:
  - If the string returns a **positive** unique number you can **make** it **negative** by **adding "-"** as a **second parameter**
  - If the string returns a **negative** unique number you can **make** it **positive** by **adding "+"** as a **second parameter**
  - If your **satisfied** with the **unique number** then you **don't need** to put a **second parameter**

- **Numbers**
  - **Number Methods**
    - **To String**
      - The string will return as a number

```
<j-number data="x" data-number-tostring></j-number>
```

```
<p id="numTest">Number: </p>
  <jh-script>
    <j-let data="x = 123" data-let></j-let>

    <j-append element="'#numTest'" data-append-block>
      <j-value-rbr/>
        <j-number data="x" data-number-tostring>
         </j-number><j-value-br/>
        <j-number data="123" data-number-tostring>
         </j-number><j-value-br/>
        <j-number data="123 + 23" data-number-tostring>
         </j-number>
    </j-append>
  </jh-script>
```

    - **To Exponential**
      - The string will return as exponential notation

```
<j-number data="x" method="number" type="toExponential"
param data-number></j-number>
```

- **To Exponential**
  - Here is the working code of the preview:

```
<p id="numTest">Number: </p>
<jh-script>
  <j-let data="x = 9.656" data-let></j-let>

  <j-append element="'#numTest'" data-append-block>
    <j-value-rbr/>
      <j-number data="x" method="number" type="toExponential"
        param data-number></j-number><j-value-br/>
      <j-number data="x" method="number" type="toExponential"
        param="2" data-number></j-number><j-value-br/>
      <j-number data="x" method="number" type="toExponential"
        param="4" data-number></j-number><j-value-br/>
      <j-number data="x" method="number" type="toExponential"
        param="6" data-number></j-number>
  </j-append>
</jh-script>
```

- **To Fixed**
  - The string will return as a number of decimals

```
<j-number data="x" method="number" type="toFixed" param="0"
data-number></j-number>
```

```
<p id="numTest">Number: </p>
<jh-script>
    <j-let data="x = 9.656" data-let></j-let>
    <j-append element="'#numTest'" data-append-block>
      <j-value-rbr/>
        <j-number data="x" method="number" type="toFixed"
          param="0" data-number></j-number><j-value-br/>
        <j-number data="x" method="number" type="toFixed"
          param="2" data-number></j-number><j-value-br/>
        <j-number data="x" method="number" type="toFixed"
          param="4" data-number></j-number><j-value-br/>
        <j-number data="x" method="number" type="toFixed"
          param="6" data-number></j-number>
    </j-append>
</jh-script>
```

- **To Precision**
  - The string returns the number with a specific length

```
<j-number data="x" method="number" type="toPrecision" param
data-number></j-number>
```

```
<p id="numTest">Number: </p>
<jh-script>
    <j-let data="x = 9.656" data-let></j-let>

    <j-append element="'#numTest'" data-append-block>
      <j-value-rbr/>
        <j-number data="x" method="number"
          type="toPrecision" param data-number>
          </j-number><j-value-br/>
        <j-number data="x" method="number"
          type="toPrecision" param="2" data-number>
          </j-number><j-value-br/>
        <j-number data="x" method="number"
          type="toPrecision" param="4" data-number>
          </j-number><j-value-br/>
        <j-number data="x" method="number"
          type="toPrecision" param="6" data-number>
          </j-number>
    </j-append>
</jh-script>
```

- **Value Of**
  - The number returns as itself which is a number

```
<j-number data="x" data-number-valueof></j-number>
```

- **Value Of**
  - Here is the working code of the preview:

```
<p id="numTest">Number: </p>
<jh-script>
<j-let data="x = 123" data-let></j-let>

<j-append element="'#numTest'" data-append-block>
<j-value-rbr/>
<j-number data="x" data-number-valueof></j-number><j-value-br/>
<j-number data="123" data-number-valueof></j-number><j-value-br/>
<j-number data="100 + 23" data-number-valueof></j-number>
</j-append>
</jh-script>
```

- **Number**
  - Converts variables to numbers

```
<j-number><j-value-true/></j-number><j-value-br/>
```

  - **For example**:
    - `Number(true);`
    - `Number(false);`
    - `Number("10");`
    - `Number("  10");`
    - `Number("10  ");`
    - `Number(" 10  ");`
    - `Number("10.33");`
    - `Number("10,33");`
    - `Number("10 33");`
    - `Number("John");`

- **Number**
  - Here is the working code of the preview:

```
<p id="numTest">Number: </p>
  <jh-script>
    <j-append element="'#numTest'" data-append-block>
      <j-value-rbr/>
        <j-number><j-value-true/></j-number><j-value-br/>
        <j-number><j-value-false/></j-number><j-value-br/>
        <j-number><j-value data='"10"' data-value></j-value>
         </j-number><j-value-br/>
        <j-number><j-value data='"      10"' data-value>
         </j-value></j-number><j-value-br/>
        <j-number><j-value data='"10      "' data-value>
         </j-value></j-number><j-value-br/>
        <j-number><j-value data='"   10   "' data-value>
         </j-value></j-number><j-value-br/>
        <j-number><j-value data='"10.33"' data-value>
         </j-value></j-number><j-value-br/>
        <j-number><j-value data='"10,33"' data-value>
         </j-value></j-number><j-value-br/>
        <j-number><j-value data='"10 33"' data-value>
         </j-value></j-number><j-value-br/>
        <j-number><j-value data='"John"' data-value>
         </j-value></j-number>
    </j-append>
  </jh-script>
```

- **Parse Int**
  - Parse variables to a whole number, number first before letters with spaces are allowed

```
<j-parse-int><j-value data='"-10"' data-value></j-value>
</j-parse-int>
```

- **Parse Int**
  - Here is the working code of the preview

```
<p id="numTest">Number: </p>
  <jh-script>
  <j-append element="'#numTest'" data-append-block>
   <j-value-rbr/>
    <j-parse-int><j-value data='"-10"' data-value></j-value>
     </j-parse-int><j-value-br/>
    <j-parse-int><j-value data='"-10.33"' data-value></j-value>
     </j-parse-int><j-value-br/>
    <j-parse-int><j-value data='"10"' data-value></j-value>
     </j-parse-int><j-value-br/>
    <j-parse-int><j-value data='"10.33"' data-value></j-value>
     </j-parse-int><j-value-br/>
    <j-parse-int><j-value data='"10 20 30"' data-value>
     </j-value></j-parse-int><j-value-br/>
    <j-parse-int><j-value data='"10 years"' data-value>
     </j-value></j-parse-int><j-value-br/>
    <j-parse-int><j-value data='"years 10"' data-value>
      </j-value></j-parse-int>
   </j-append>
  </jh-script>
```

- **Parse Float**
  - Parse variables to a number either has a decimal or not, number first before letters with spaces are allowed

```
<j-parse-float><j-value data='"10.33"' data-value></j-value>
</j-parse-float>
```

  - **For example**:
    - `parseFloat("10");`
    - `parseFloat("10.33");`
    - `parseFloat("10 20 30");`
    - `parseFloat("10 years");`
    - `parseFloat("years 10");`

- **Parse Float**
  - Here is the working code of the preview

```
<p id="numTest">Number: </p>
  <jh-script>
  <j-append element="'#numTest'" data-append-block>
   <j-value-rbr/>
    <j-parse-float><j-value data='"10"' data-value></j-value>
     </j-parse-float><j-value-br/>
    <j-parse-float><j-value data='"10.33"' data-value>
     </j-value></j-parse-float><j-value-br/>
    <j-parse-float><j-value data='"10 20 30"' data-value>
     </j-value></j-parse-float><j-value-br/>
    <j-parse-float><j-value data='"10 years"' data-value>
     </j-value></j-parse-float><j-value-br/>
    <j-parse-float><j-value data='"years 10"' data-value>
     </j-value></j-parse-float>
   </j-append>
  </jh-script>
```

- **Is Integer**
  - Check if the argument is an integer

```
<j-number method="literal" type="isInteger" param="10" data-
number></j-number>
```

```
<p id="numTest">Number: </p>
  <jh-script>

  <j-append element="'#numTest'" data-append-block>
   <j-value-rbr/>
    <j-number method="literal" type="isInteger"
      param="10" data-number></j-number><j-value-br/>
    <j-number method="literal" type="isInteger"
      param="10.5" data-number></j-number>
   </j-append>

  </jh-script>
```

- **Is Safe Integer**
  - Check if the argument is a safe integer

```
<j-number method="literal" type="isSafeInteger" param="10"
data-number></j-number>
```

```
<p id="numTest">Number is Safe Integer: </p>
  <jh-script>

  <j-append element="'#numTest'" data-append-block>
   <j-value-rbr/>
    <j-number method="literal" type="isSafeInteger"
      param="10" data-number></j-number><j-value-br/>
    <j-number method="literal" type="isSafeInteger"
      param="12345678901234567890" data-number></j-number>
   </j-append>
  </jh-script>
```

- **Number Parse Float**
  - Another way to convert string to number either has a decimal or not

  - **For example**:
    - `Number.parseFloat("10");`
    - `Number.parseFloat("10.33");`
    - `Number.parseFloat("10 20 30");`
    - `Number.parseFloat("10 years");`
    - `Number.parseFloat("years 10");`

- **Number Parse Float**
  - Here is the working code of the preview

```
<p id="numTest">Number.parseFloat(): </p>
  <jh-script>

  <j-append element="'#numTest'" data-append-block>
   <j-value-rbr/>
    <j-number method="literal" type="parseFloat" param="'10'"
      data-number></j-number><j-value-br/>
   <j-number method="literal" type="parseFloat"
     param="'10.33'" data-number></j-number><j-value-br/>
   <j-number method="literal" type="parseFloat"
     param="'10 20 30'" data-number></j-number><j-value-br/>
   <j-number method="literal" type="parseFloat"
     param="'10 years'" data-number></j-number><j-value-br/>
    <j-number method="literal" type="parseFloat"
      param="'year 10'" data-number></j-number>
   </j-append>
  </jh-script>
```

- **Number Parse Int**
  - Another way to convert string to whole number

```
<j-number method="literal" type="parseInt" param="'-10'" data-
number></j-number>
```

  - **For example:**
    - `Number.parseInt("-10");`
    - `Number.parseInt("-10.33");`
    - `Number.parseInt("10");`
    - `Number.parseInt("10.33");`
    - `Number.parseInt("10 20 30");`
    - `Number.parseInt("10 years");`
    - `Number.parseInt("years 10");`

- **Number Parse Int**
  - Here is the working code of the preview:

```
<p id="numTest">Number.parseFloat(): </p>
<jh-script>

  <j-append element="'#numTest'" data-append-block>
   <j-value-rbr/>
    <j-number method="literal" type="parseInt" param="'-10'"
      data-number></j-number><j-value-br/>
  <j-number method="literal" type="parseInt" param="'-10.33'"
    data-number></j-number><j-value-br/>
  <j-number method="literal" type="parseInt" param="'10'"
    data-number></j-number><j-value-br/>
   <j-number method="literal" type="parseInt" param="'10.33'"
     data-number></j-number><j-value-br/>
   <j-number method="literal" type="parseInt" param="'10 20 30'"
     data-number></j-number><j-value-br/>
   <j-number method="literal" type="parseInt" param="'10 years'"
     data-number></j-number><j-value-br/>
    <j-number method="literal" type="parseInt" param="'year 10'"
      data-number></j-number>
   </j-append>
</jh-script>
```

- **Properties**
  - **EPSILON**
    - Between 1 and the smallest number of its difference
  - **MAX VALUE**
    - The max number possible
  - **MIN VALUE**
    - The minimum number possible
  - **MAX SAFE VALUE**
    - The max safe integer possible
  - **MIN SAFE VALUE**
    - The min safe integer possible
  - **POSITIVE INFINITY**
    - Literally the value of infinity
  - **NEGATIVE INFINITY**
    - Literally the value of negative infinity
  - **NOT A NUMBER**
    - Literally the value of a not number

- **Properties**
  - Here is the preview and code of the listed number properties:

```
<p id="numTest"><span style="color:red;">Number Property:</span> </p>
  <jh-script>

  <j-append element="'#numTest'" data-append-block>
   <j-value-rbr/>
    <j-value data="'<b>EPSILON: </b>'" data-value></j-value><j-value-add/>
     <j-number method="property" type="EPSILON" data-number>
      </j-number><j-value-br/>
    <j-value data="'<b>MAX_VALUE: </b>'" data-value></j-value><j-value-add/>
     <j-number method="property" type="MAX_VALUE" data-number>
      </j-number><j-value-br/>
    <j-value data="'<b>MIN_VALUE: </b>'" data-value></j-value><j-value-add/>
     <j-number method="property" type="MIN_VALUE" data-number>
      </j-number><j-value-br/>
    <j-value data="'<b>MAX_SAFE_INTEGER: </b>'" data-value></j-value>
     <j-value-add/>
     <j-number method="property" type="MAX_SAFE_INTEGER" data-number>
      </j-number><j-value-br/>
    <j-value data="'<b>MIN_SAFE_INTEGER: </b>'" data-value></j-value>
     <j-value-add/>
     <j-number method="property" type="MIN_SAFE_INTEGER" data-number>
      </j-number><j-value-br/>
    <j-value data="'<b>POSITIVE_INFINITY: </b>'" data-value></j-value>
     <j-value-add/>
     <j-number method="property" type="POSITIVE_INFINITY" data-number>
      </j-number><j-value-br/>
    <j-value data="'<b>NEGATIVE_INFINITY: </b>'" data-value></j-value>
     <j-value-add/>
     <j-number method="property" type="NEGATIVE_INFINITY" data-number>
       </j-number><j-value-br/>
    <j-value data="'<b>NOT A NUMBER: </b>'" data-value></j-value>
      <j-value-add/>
     <j-number method="property" type="NaN" data-number></j-number>
   </j-append>
  </jh-script>
```

- **Extended Methods**
  - **Count Number**
    - Literally counts numbers as string or numbers, and count decimal places if allowed

```
<j-number method="jaihoe" type="countNum"
  param="57.25" data-number-method></j-number>
```

```
<p style="margin:0;" id="numTest">
 <span>Number Count:<br/></span>
</p>
  <jh-script>
   <j-append element="'#numTest'" data-append-block>
     <j-number method="jaihoe" type="countNum"
       param="57.25" data-number-method></j-number>
     <j-value-br/>
    <j-number method="jaihoe" type="countNum"
       param="72.45,true" data-number-method></j-number>

    <j-value-br/>
    <j-number method="jaihoe" type="countNum"
      param="'23,4567.25'" data-number-method></j-number>
    <j-value-br/>
    <j-number method="jaihoe" type="countNum"
      param="'23,4567.25','true'" data-number-method></j-number>

   </j-append>
</jh-script>
```

- **Number Form**
  - Converts number into a specific form, depending on the delimiter

```
<j-number method="jaihoe" type="numForm"
  param="12358.25" data-number-method></j-number>
```

- **Number Form**
  - Here is the preview and the working code below:

```
<p style="margin:0;" id="numTest"><b>Number Form:<br/></b></p>
<jh-script>
 <j-append element="'#numTest'" data-append-block>
  <j-number method="jaihoe" type="numForm"
    param="12358.25" data-number-method></j-number>
    <j-value-br/>
  <j-number method="jaihoe" type="numForm"
    param="'15,753.25','r'" data-number-method></j-number>
    <j-value-br/>
  <j-number method="jaihoe" type="numForm"
    param="'25,758.25','w'" data-number-method></j-number>
    <j-value-br/>
  <j-number method="jaihoe" type="numForm"
    param="1234,'rn'" data-number-method></j-number>
    <j-value-br/>
  <j-number method="jaihoe" type="numForm"
    param="5678,'expand'" data-number-method></j-number>
 </j-append>
</jh-script>
```

  - **For example:**
    - **12358.25** is converted to **standard form**, add **'s'** as a second parameter or **not** to make it **12,358.25**
    - **'15,753.25'** is converted to real number, add **'r'** as a second parameter to make it **15753.25**
    - **'25,758.25'** is converted to words, add **'w'** as a second parameter to make it **"Twenty Five Thousand Seven Hundred Fifty Eight Point Two Five"**
    - **1234** is converted to roman numerals, add **'rn'** as a second parameter to make it "**MCCXXXIV"**
    - **5678** is converted to **expanded form**, add **'expand'** as a second parameter to make it **"5000+600+70+8"**

- **Is Whole Number**
  - It returns a boolean value if the number is a whole number or not

```
<j-number method="jaihoe" type="isWholeNum"
  param="72" data-number-method></j-number>
```

```
<p style="margin:0;" id="numTest"><b>Whole Number?:<br/></b></p>
<jh-script>
 <j-append element="'#numTest'" data-append-block>
   <j-number method="jaihoe" type="isWholeNum"
     param="'25.00'" data-number-method></j-number><j-value-br/>
    <j-number method="jaihoe" type="isWholeNum"
      param="62.25" data-number-method></j-number><j-value-br/>
    <j-number method="jaihoe" type="isWholeNum"
      param="72" data-number-method></j-number>
  </j-append>
</jh-script>
```

- **Is Decimal Number**
  - It returns a boolean value if the number is a decimal number or not

```
<j-number method="jaihoe" type="isDecimalNum"
  param="'25.15'" data-number-method></j-number>
```

- **Is Decimal Number**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="numTest"><b>Decimal Number?:<br/></b></p>
<jh-script>
   <j-append element="'#numTest'" data-append-block>
   <j-number method="jaihoe" type="isDecimalNum"
     param="'25.15'" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isDecimalNum"
     param="'62.00'" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isDecimalNum"
     param="72.01" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isDecimalNum"
     param="75" data-number-method></j-number>
   </j-append>
</jh-script>
```

- **Is Odd Number**
  - It returns a boolean value if the number is an odd number or not

```
<j-number method="jaihoe" type="isOdd"
  param="'5'" data-number-method></j-number>
```

```
<p style="margin:0;" id="numTest"><b>Odd Number?:<br/></b></p>
  <jh-script>
   <j-append element="'#numTest'" data-append-block>
   <j-number method="jaihoe" type="isOdd"
     param="'5'" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isOdd"
     param="'2'" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isOdd"
     param="3" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isOdd"
     param="4" data-number-method></j-number>
   </j-append>
  </jh-script>
```

- **Is Even Number**
  - It returns a boolean value if the number is an even number or not

```
<j-number method="jaihoe" type="isEven"
     param="'5'" data-number-method></j-number><j-value-br/>
```

```
<p style="margin:0;" id="numTest"><b>Even Number?:<br/></b></p>
  <jh-script>
   <j-append element="'#numTest'" data-append-block>
   <j-number method="jaihoe" type="isEven"
     param="'5'" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isEven"
     param="'2'" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isEven"
     param="3" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isEven"
     param="4" data-number-method></j-number>
   </j-append>
  </jh-script>
```

- **Is Positive Number**
  - It returns a boolean value if the number is a positive number or not

```
<j-number method="jaihoe" type="isPositive"
  param="'5'" data-number-method></j-number>
```

- **Is Positive Number**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="numTest"><b>Positive Number?:<br/></b></p>
  <jh-script>
   <j-append element="'#numTest'" data-append-block>
   <j-number method="jaihoe" type="isPositive"
     param="'5'" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isPositive"
     param="6" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isPositive"
     param="'-7'" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isPositive"
     param="-8" data-number-method></j-number>
   </j-append>
  </jh-script>
```

- **Is Negative Number**
  - It returns a boolean value if the number is a negative number or not

```
<j-number method="jaihoe" type="isNegative"
  param="'5'" data-number-method></j-number>
```

```
<p style="margin:0;" id="numTest"><b>Negative Number?:<br/></b></p>
  <jh-script>
   <j-append element="'#numTest'" data-append-block>
   <j-number method="jaihoe" type="isNegative"
     param="'5'" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isNegative"
     param="6" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isNegative"
     param="'-7'" data-number-method></j-number><j-value-br/>
   <j-number method="jaihoe" type="isNegative"
     param="-8" data-number-method></j-number>
   </j-append>
  </jh-script>
```

- **Array**
  - **Default Set**
    - This is how you create an array:
      - **Default Implementation**
        - This is best for open values or variables

```
<p id="arrayTest">
<span style="color:red;">Array Set:</span>
</p>
  <jh-script>
  <j-let data="odd" data-eq data-let>
    <j-array>
      <j-arel data="1" data-arel></j-arel>
      <j-arel data="3" data-arel></j-arel>
      <j-arel data="5" data-arel-close></j-arel>
    </j-array>
  </j-let>

  <j-append element="'#arrayTest'" data-append-block>
    <j-array data="odd" data-array-tostring></j-array>
   </j-append>
  </jh-script>
```

      - **Single Quote Implementation**
        - This is best for string that use double quote values

- **Single Quote Implementation**
  - Here is the working code of the preview:

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
  <j-let data="quote" data-eq data-let>
    <j-array>
      <j-arel data-sq='<br>"This is good." -KPC' data-arel-sq>
       </j-arel>
      <j-arel data-sq='<br>"Be the star!" -JFC' data-arel-sq>
       </j-arel>
      <j-arel data-sq='<br>"Hot or Cold?" -KP' data-arel-sq-close>
       </j-arel>
    </j-array>
  </j-let>

  <j-append element="'#arrayTest'" data-append-block>
    <j-array data="quote" data-array-tostring></j-array>
   </j-append>
  </jh-script>
```

- **Double Quote Implementation**
  - This is best for string that use single quote values

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
  <j-let data="greet" data-eq data-let>
    <j-array>
      <j-arel data-dq="<br>I'm Joe" data-arel-dq></j-arel>
      <j-arel data-dq="<br>I'm Jim" data-arel-dq></j-arel>
      <j-arel data-dq="<br>I'm John" data-arel-dq-close></j-arel>
    </j-array>
  </j-let>

  <j-append element="'#arrayTest'" data-append-block>
    <j-array data="greet" data-array-tostring></j-array>
   </j-append>
  </jh-script>
```

- **Separate Set**
  - You can declare the array then set the values manually

```
<j-let data="even" data-litarr data-let></j-let>
```

    - That is equals to **let even = [];**

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
   <j-let data="even" data-litarr data-let></j-let>

   <j-set data="even[0] = 2" data-set></j-set>
   <j-set data="even[2] = 6" data-set></j-set>
   <j-set data="even[1] = 4" data-set></j-set>

   <j-append element="'#arrayTest'" data-append-block>
    <j-array data="even" data-array-tostring></j-array>
   </j-append>
  </jh-script>
```

  - You can replace **"Array to String"** method to **"Full Access Array"**

```
<j-value data="even" data-value></j-value>
```

  - You can replace **"hardcode array set"** to **"data key set"** method

```
<j-set data="even" data-key="0" data-key-eq="2" data-set></j-set>
<j-set data="even" data-key="2" data-key-eq="6" data-set></j-set>
<j-set data="even" data-key="1" data-key-eq="4" data-set></j-set>
```

  - **All together**

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
<jh-script>
   <j-let data="even" data-litarr data-let></j-let>

<j-set data="even" data-key="0" data-key-eq="2" data-set></j-set>
<j-set data="even" data-key="2" data-key-eq="6" data-set></j-set>
<j-set data="even" data-key="1" data-key-eq="4" data-set></j-set>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="even" data-value></j-value>
   </j-append>
</jh-script>
```

- **Array Methods**
  - **Length**
    - Returns the length of the array

```
<j-array data="fruits" data-array-length></j-array>
```

```
<p id="arrayTest">
 <span style="color:red;">Array Length:</span>
</p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-append element="'#arrayTest'" data-append-block>
    <j-array data="fruits" data-array-length></j-array>
   </j-append>
  </jh-script>
```

  - **Is Array**
    - Returns if the variable is an array

```
<j-array method="literal" type="isArray" param="fruits"
data-array-method></j-array>
```

- **Is Array**
  - Here is the working code of the preview:

```
<p id="arrayTest"><span style="color:red;">Is Array:</span> </p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-append element="'#arrayTest'" data-append-block>
    <j-array method="literal" type="isArray" param="fruits"
      data-array-method></j-array>
   </j-append>
  </jh-script>
```

- **Array to String**
  - Literally converts array to string

```
<j-array data="fruits" data-array-tostring></j-array>
```

```
<p id="arrayTest">
 <span style="color:red;">Array Set:</span>
</p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-append element="'#arrayTest'" data-append-block>
    <j-array data="fruits" data-array-tostring></j-array>
   </j-append>
  </jh-script>
```

- **Join**
  - Array are joined together as a string, it act as a separator to elements of array defined

```
<j-array data="fruits" method="array" type="join" param="' * '"
  data-array-method></j-array>
```

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-append element="'#arrayTest'" data-append-block>
    <j-array data="fruits" method="array" type="join"
      param="' * '" data-array-method></j-array>
   </j-append>
  </jh-script>
```

- **Pop**
  - Removes the last element of the array

```
<j-array-set data="fruits" method="array" type="pop" param
  data-array-method></j-array-set>
```

  - The difference between **j-array** and **j-array-set** is **j-array-set** is used for **non-nested-dependent** blocks.

- **Pop**
  - Here is the working code of the preview:

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>
   <j-array-set data="fruits" method="array" type="pop" param
     data-array-method></j-array-set>
   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="fruits" data-value></j-value>
   </j-append>
  </jh-script>
```

- **Push**
  - Adds a new element at the end of the array

```
<j-array-set data="fruits" method="array" type="push" param="'Kiwi'"
  data-array-method></j-array-set>
```

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>
   <j-array-set data="fruits" method="array" type="push"
     param="'Kiwi'" data-array-method></j-array-set>
   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="fruits" data-value></j-value>
   </j-append>
  </jh-script>
```

- **Shift**
  - Removes the first element of the array and other elements are pushed to the left

```
<j-array-set data="fruits" data-array-shift></j-array-set>
```

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>
   <j-array-set data="fruits" data-array-shift></j-array-set>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="fruits" data-value></j-value>
   </j-append>
  </jh-script>
```

- **Unshift**
  - Appends an element to the beginning of the array

```
<j-array-set data="fruits" method="array" type="unshift"
  param="'Lemon'" data-array-method></j-array-set>
```

  - As you can see, “Lemon” was added to the array set

- **Unshift**
  - Here is the working code of the preview:

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>
   <j-array-set data="fruits" method="array" type="unshift"
     param="'Lemon'" data-array-method></j-array-set>
   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="fruits" data-value></j-value>
   </j-append>
  </jh-script>
```

- **Delete**
  - Literally deletes the element of the array leaving it undefined

```
<j-array-del data="fruits" data-key="0" data-array-del></j-array-del>
```

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
<jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
    </j-array>
   </j-let>
   <j-append element="'#arrayTest'" data-append-block>
  <j-value data="'<br>The first fruit is: '+fruits[0]" data-value>
    </j-value></j-append>
   <j-array-del data="fruits" data-key="0" data-array-del>
    </j-array-del>
   <j-append element="'#arrayTest'" data-append-block>
   <j-value data="'<br>The first fruit is: '+fruits[0]" data-value>
     </j-value></j-append>
</jh-script>
```

- **Concat**
  - Appends an array set or item to an existing array set

```
<j-array data="oldClassmates" method="array" type="concat"
  param="newStudents,exhangeStudents" data-array-method></j-array>
```

```
<p id="arrayTest"><span style="color:red;">Class 1:</span> </p>
  <jh-script>
   <j-let data="oldClassmates" data-eq data-let>
    <j-array>
      <j-arel data-dq="John" data-arel-dq></j-arel>
      <j-arel data-dq="James" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-let data="newStudents" data-eq data-let>
    <j-array>
      <j-arel data-dq="Jules" data-arel-dq></j-arel>
      <j-arel data-dq="Jill" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-let data="exhangeStudents" data-eq data-let>
    <j-array>
      <j-arel data-dq="Jake" data-arel-dq></j-arel>
      <j-arel data-dq="Joa" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-let data="classA" data-eq data-let>
    <j-array data="oldClassmates" method="array" type="concat"
     param="newStudents,exhangeStudents" data-array-method>
      </j-array>
   </j-let>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="classA" data-value></j-value>
   </j-append>
  </jh-script>
```

- **Splice**
  - Adds a group or a single array element to a specific position and removes a certain range of an array element

```
<j-let data="removed" data-eq data-let>
  <j-array data="fruits" method="array" type="splice"
    param="2,2,'Lemon','Kiwi'" data-array-method></j-array>
</j-let>
```

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value-rbr/>
    <j-value data="'<b>Original Array:</b> '+fruits"
      data-value></j-value>
     <j-value-lbr/>
   </j-append>

   <j-let data="removed" data-eq data-let>
    <j-array data="fruits" method="array" type="splice"
      param="2,2,'Lemon','Kiwi'" data-array-method></j-array>
   </j-let>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="'<b>New Array:</b> '+fruits" data-value>
      </j-value><j-value-br/>
   <j-value data="'<b>Removed:</b> '+removed" data-value>
     </j-value>
   </j-append>
  </jh-script>
```

- **Slice**
  - Literally slice the array set into a new array

```
<j-let data="citrus" data-eq data-let>
  <j-array data="fruits" method="array" type="slice"
    param="3" data-array-method></j-array>
</j-let>
```

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Lemon" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-let data="citrus" data-eq data-let>
    <j-array data="fruits" method="array" type="slice"
      param="3" data-array-method></j-array>
   </j-let>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value-rbr/>
    <j-value data="'<b>Fruits:</b> '+fruits"
      data-value></j-value><j-value-br/>
    <j-value data="'<b>Citrus Fruits:</b> '+citrus"
      data-value>
      </j-value>
   </j-append>
  </jh-script>
```

- **Sort**
  - Literally sorts the array set in ascending order (each elements) and allows other kinds of sorting depending on the delimiter

```
<j-array-set data="fruits" method="array" type="sort"
  param data-array-method></j-array-set>
```

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
<jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Lemon" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-array-set data="fruits" method="array" type="sort"
      param data-array-method></j-array-set>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="fruits" data-value></j-value>
   </j-append>
</jh-script>
```

- **Reverse**
  - Literally reverse the order of the array set (each elements)

```
<j-array-set data="fruits" method="array" type="reverse"
      param data-array-method></j-array-set>
```

  - **From:** Apple, Banana, Lemon, Mango, Orang

- **Reverse**
  - Here is the working code of the preview:

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Banana" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Lemon" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-array-set data="fruits" method="array" type="sort"
      param data-array-method></j-array-set>
   <j-array-set data="fruits" method="array" type="reverse"
      param data-array-method></j-array-set>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="fruits" data-value></j-value>
   </j-append>


  </jh-script>
```

- **Map**
  - Creates a new array that performs the action of the specified function

```
<j-const data="numbers2" data-eq data-const>
  <j-array data="numbers1" method="array" type="map"
    param="myFunction" data-array-method></j-array>
</j-const>
```

  - For example: If **45, 4, 9, 16, 25** are multiplied by **2**

- **Map**
  - Here is the working code of the preview

```
<p id="arrayTest"><span style="color:red;">Array Set:</span> </p>
<jh-script>
   <j-const data="numbers1" data-eq data-const>
    <j-array>
      <j-arel data="45" data-arel></j-arel>
      <j-arel data="4" data-arel></j-arel>
      <j-arel data="9" data-arel></j-arel>
      <j-arel data="16" data-arel></j-arel>
      <j-arel data="25" data-arel-close></j-arel>
    </j-array>
   </j-const>

   <j-const data="numbers2" data-eq data-const>
    <j-array data="numbers1" method="array" type="map"
      param="myFunction" data-array-method></j-array>
   </j-const>

   <j-fx name="myFunction" param="value " data-fx>
    <j-return value="value * 2" data-return></j-return>
   </j-fx>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="numbers2" data-value></j-value>
   </j-append>
</jh-script>
```

- **Filter**
  - Creates a new array that satisfies the certain condition

```
<j-const data="over18" data-eq data-const>
  <j-array data="numbers1" method="array" type="filter"
    param="myFunction" data-array-method></j-array>
</j-const>
```

- **Filter**
  - Here is the working code of the preview:

```
<p id="arrayTest">
 <span style="color:red;">Age over (18):</span>
</p>
<jh-script>
   <j-const data="ages" data-eq data-const>
    <j-array>
      <j-arel data="18" data-arel></j-arel>
      <j-arel data="25" data-arel></j-arel>
      <j-arel data="9" data-arel></j-arel>
      <j-arel data="16" data-arel></j-arel>
      <j-arel data="45" data-arel-close></j-arel>
    </j-array>
   </j-const>

   <j-const data="over18" data-eq data-const>
    <j-array data="ages" method="array" type="filter"
      param="myFunction" data-array-method></j-array>
   </j-const>
   <j-fx name="myFunction" param="value" data-fx>
    <j-return value="value > 17" data-return></j-return>
   </j-fx>
   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="over18" data-value></j-value>
   </j-append>
</jh-script>
```

- **Reduce**
  - It works like map that runs a function for each array element and it does not actually reduce elements in the array

```
<j-let data="sum" data-eq data-let>
  <j-array data="numbers" method="array" type="reduce"
    param="myFunction" data-array-method></j-array>
</j-let>
```

  - The sum of **45, 4, 9, 16, 25** is **99**

- **Reduce**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="arrayTest">
   <span style="color:red;">The sum of array is:</span>
</p>
<jh-script>
      <j-const data="numbers" data-eq data-const>
       <j-array>
         <j-arel data="45" data-arel></j-arel>
         <j-arel data="4" data-arel></j-arel>
         <j-arel data="9" data-arel></j-arel>
         <j-arel data="16" data-arel></j-arel>
         <j-arel data="25" data-arel-close></j-arel>
       </j-array>
      </j-const>

      <j-let data="sum" data-eq data-let>
       <j-array data="numbers" method="array" type="reduce"
         param="myFunction" data-array-method></j-array>
      </j-let>
      <j-fx name="myFunction" param="total, value" data-fx>
       <j-return value="total + value" data-return></j-return>
      </j-fx>
      <j-append element="'#arrayTest'" data-append-block>
       <j-value data="sum" data-value></j-value>
      </j-append>
</jh-script>
```

- **Reduce Right**
  - It works like reduce but the function runs from right to left in the array and it is not also reduce the elements of the array

```
<j-let data="sum" data-eq data-let>
  <j-array data="numbers" method="array" type="reduceRight"
    param="myFunction" data-array-method></j-array>
</j-let>
```

  - The sum of **10, 15, 20, 25, 30** is **100**

- **Reduce Right**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="arrayTest">
    <span style="color:red;">The sum of array is:</span>
 </p>
 <jh-script>
       <j-const data="numbers" data-eq data-const>
        <j-array>
          <j-arel data="30" data-arel></j-arel>
          <j-arel data="25" data-arel></j-arel>
          <j-arel data="20" data-arel></j-arel>
          <j-arel data="15" data-arel></j-arel>
          <j-arel data="10" data-arel-close></j-arel>
        </j-array>
       </j-const>

       <j-let data="sum" data-eq data-let>
        <j-array data="numbers" method="array" type="reduceRight"
          param="myFunction" data-array-method></j-array>
       </j-let>
       <j-fx name="myFunction" param="total, value" data-fx>
        <j-return value="total + value" data-return></j-return>
       </j-fx>

       <j-append element="'#arrayTest'" data-append-block>
        <j-value data="sum" data-value></j-value>
       </j-append>
 </jh-script>
```

- **Every**
  - Checks if all array elements pass the condition

```
<j-let data="canGoToBar" data-eq data-let>
  <j-array data="ages" method="array" type="every"
    param="myFunction" data-array-method></j-array>
</j-let>
```

  - All ages **45, 26, 22, 23, 25** can go to the bar

- **Every**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="arrayTest">
    <span style="color:red;">All can go to bar (22 up):</span>
 </p>
 <jh-script>
       <j-const data="ages" data-eq data-const>
        <j-array>
          <j-arel data="45" data-arel></j-arel>
          <j-arel data="26" data-arel></j-arel>
          <j-arel data="22" data-arel></j-arel>
          <j-arel data="23" data-arel></j-arel>
          <j-arel data="25" data-arel-close></j-arel>
        </j-array>
       </j-const>

       <j-let data="canGoToBar" data-eq data-let>
        <j-array data="ages" method="array" type="every"
          param="myFunction" data-array-method></j-array>
       </j-let>
       <j-fx name="myFunction" param="value" data-fx>
        <j-return value="value >= 22" data-return></j-return>
       </j-fx>
       <j-append element="'#arrayTest'" data-append-block>
        <j-value data="canGoToBar" data-value></j-value>
       </j-append>
 </jh-script>
```

- **Some**
  - Checks if some array elements pass the condition

```
<j-let data="canGoToBar" data-eq data-let>
  <j-array data="ages" method="array" type="some"
    param="myFunction" data-array-method></j-array>
</j-let>
```

  - From ages **45,** 4**,** 9**,** 16**, 25** there are some people can go to **bar**

- **Some**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="arrayTest">
    <span style="color:red;">Some can go to bar (22 up):</span>
 </p>
 <jh-script>
       <j-const data="ages" data-eq data-const>
        <j-array>
          <j-arel data="45" data-arel></j-arel>
          <j-arel data="4" data-arel></j-arel>
          <j-arel data="9" data-arel></j-arel>
          <j-arel data="16" data-arel></j-arel>
          <j-arel data="25" data-arel-close></j-arel>
        </j-array>
       </j-const>

       <j-let data="canGoToBar" data-eq data-let>
        <j-array data="ages" method="array" type="some"
          param="myFunction" data-array-method></j-array>
       </j-let>
       <j-fx name="myFunction" param="value" data-fx>
        <j-return value="value >= 22" data-return></j-return>
       </j-fx>
       <j-append element="'#arrayTest'" data-append-block>
        <j-value data="canGoToBar" data-value></j-value>
       </j-append>
 </jh-script>
```

- **IndexOf**
  - Searches and returns the position of the array element

```
<j-let data="position" data-eq data-let>
    <j-array data="fruits" method="array" type="indexOf"
      param="'Apple'" data-extend-method="+1"
      data-array-extend></j-array>
   </j-let>
```

  - So the fruits are listed as (1) Apple, (2) Orange, (3) Apple, (4) Mango in **normal** counting
  - So the fruits are listed as (0) Apple, (1) Orange, (2) Apple, (3) Mango in **array** counting

- **IndexOf**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="arrayTest">
  <span style="color:red;">Position of </span><b>(Apple):</b>
</p>

<p style="margin:0;" id="arrayTest2">
  <span style="color:red;">Position of </span><b>(Mango):</b>
</p>

<jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-let data="position" data-eq data-let>
    <j-array data="fruits" method="array" type="indexOf"
      param="'Apple'" data-extend-method="+1"
      data-array-extend></j-array>
   </j-let>

   <j-let data="position2" data-eq data-let>
    <j-array data="fruits" method="array" type="indexOf"
      param="'Mango'" data-extend-method="+1"
      data-array-extend></j-array>
   </j-let>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="position" data-value></j-value>
   </j-append>

   <j-append element="'#arrayTest2'" data-append-block>
    <j-value data="position2" data-value></j-value>
   </j-append>

</jh-script>
```

  - As you can see the method is **extended**, because if not it will only return the array counting value which is 0, if the method is added to one then it will return the normal counting value which is 1 for Apple.

  - Same goes with the **other elements** in the array.

- **Last IndexOf**
  - Similar to IndexOf but it returns the last position of the array element

```
<p style="margin:0;" id="arrayTest">
 <span style="color:red;">Last Position of </span><b>(Apple):</b>
</p>
<p style="margin:0;" id="arrayTest2">
 <span style="color:red;">Last Position of </span><b>(Mango):</b>
</p>

  <jh-script>
   <j-let data="fruits" data-eq data-let>
    <j-array>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Orange" data-arel-dq></j-arel>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

   <j-let data="position" data-eq data-let>
    <j-array data="fruits" method="array" type="lastIndexOf"
      param="'Apple'" data-extend-method="+1" data-array-extend>
    </j-array>
   </j-let>
   <j-let data="position2" data-eq data-let>
    <j-array data="fruits" method="array" type="lastIndexOf"
      param="'Mango'" data-extend-method="+1" data-array-extend>
    </j-array>
   </j-let>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="position" data-value></j-value>
   </j-append>
   <j-append element="'#arrayTest2'" data-append-block>
    <j-value data="position2" data-value></j-value>
   </j-append>

  </jh-script>
```

- **Find**
  - Search for the first array element that meets the defined condition and returns its value

```
<p style="margin:0;" id="arrayTest">
 <span style="color:red;">First number over 18 is </span>
 <b>(value): </b>
</p>

<jh-script>
   <j-let data="numbers" data-eq data-let>
    <j-array>
      <j-arel data="4" data-arel></j-arel>
      <j-arel data="9" data-arel></j-arel>
      <j-arel data="16" data-arel></j-arel>
      <j-arel data="25" data-arel></j-arel>
      <j-arel data="29" data-arel-close></j-arel>
    </j-array>
   </j-let>

   <j-let data="first" data-eq data-let>
    <j-array data="numbers" method="array" type="find"
      param="myFunction" data-array-method></j-array>
   </j-let>

   <j-fx name="myFunction" param="value" data-fx>
    <j-return value="value > 18" data-return></j-return>
   </j-fx>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="first" data-value></j-value>
   </j-append>

</jh-script>
```

  - So from 4, 9, 16, 25, 29 the first person allowed to enter is aged 25

- **Find Index**
  - Search for the first array element that meets the defined condition and returns its index

```
<p style="margin:0;" id="arrayTest">
 <span style="color:red;">First number over 18 is </span>
 <b>(value): </b>
</p>

<jh-script>
   <j-let data="numbers" data-eq data-let>
    <j-array>
      <j-arel data="4" data-arel></j-arel>
      <j-arel data="9" data-arel></j-arel>
      <j-arel data="16" data-arel></j-arel>
      <j-arel data="25" data-arel></j-arel>
      <j-arel data="29" data-arel-close></j-arel>
    </j-array>
   </j-let>

   <j-let data="first" data-eq data-let>
    <j-array data="numbers" method="array" type="findIndex"
      param="myFunction" data-array-method></j-array>
   </j-let>

   <j-fx name="myFunction" param="value" data-fx>
    <j-return value="value > 18" data-return></j-return>
   </j-fx>

   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="first" data-value></j-value>
   </j-append>

</jh-script>
```

  - So from (0) 4, (1) 9, (2) 16, **(3)** 25, (4) 29 the first person allowed to enter is aged 25 which has an index of **3**

- **From**
  - Splits strings to chars and turns it to into an array

```
<j-let data="myArr" data-eq data-let>
  <j-array method="literal" type="from"
    param="'ABCDEFG'" data-array-method></j-array>
</j-let>
```

```
<p style="margin:0;" id="arrayTest">
 <span style="color:red;">Array Set: </span>
</p>
<jh-script>
   <j-let data="myArr" data-eq data-let>
    <j-array method="literal" type="from"
      param="'ABCDEFG'" data-array-method></j-array>
   </j-let>
   <j-append element="'#arrayTest'" data-append-block>
    <j-value data="myArr" data-value></j-value>
   </j-append>

</jh-script>
```

- **Keys**
  - Returns the array index of the array element

```
<j-let data="keyCollection" data-eq data-let>
  <j-array-val data="fruits" method="array" type="keys"
    param data-array-method></j-array-val>
</j-let>
```

  - The sets are:
    - (0) Banana
    - (1) Orange
    - (2) Apple
    - (3) Mango
  - Take note, use **j-array-val** to avoid complication from brackets like keys

- **Keys**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="arrayTest">
 <span style="color:red;">Array Set: </span>
</p>

<jh-script>

    <j-let data="fruits" data-eq data-let>
    <j-array>
        <j-arel data-dq="Banana" data-arel-dq></j-arel>
        <j-arel data-dq="Orange" data-arel-dq></j-arel>
        <j-arel data-dq="Apple" data-arel-dq></j-arel>
        <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
      </j-array>
    </j-let>

    <j-let data="keyCollection" data-eq data-let>
      <j-array-val data="fruits" method="array" type="keys"
        param data-array-method></j-array-val>
    </j-let>

     <j-let data="keyDisplay = ''" data-let></j-let>
     <j-let data="y = 0" data-let></j-let>
     <j-for start="let x" data-of="keyCollection" data-for>
        <j-set data="keyDisplay += x" data-set></j-set>
        <j-if data="y<3" data-if>
          <j-set data="keyDisplay += ', '" data-set></j-set>
        </j-if>
        <j-set data="y++" data-set></j-set>
    </j-for>

   <j-append element="'#arrayTest'" data-append-block>
     <j-value data="keyDisplay" data-value></j-value>
   </j-append>

</jh-script>
```

- **Entries**
  - Returns the array index and value of the array element

```
<j-let data="arrCollection" data-eq data-let>
  <j-array-val data="fruits" method="array" type="entries"
    param data-array-method></j-array-val>
</j-let>
```

```
<p style="margin:0;" id="arrayTest">
  <span style="color:red;">Array Set:</span>
</p>
<jh-script>
    <j-let data="fruits" data-eq data-let>
    <j-array>
        <j-arel data-dq="Banana" data-arel-dq></j-arel>
        <j-arel data-dq="Orange" data-arel-dq></j-arel>
        <j-arel data-dq="Apple" data-arel-dq></j-arel>
        <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
      </j-array>
    </j-let>
    <j-let data="arrCollection" data-eq data-let>
      <j-array-val data="fruits" method="array" type="entries"
        param data-array-method></j-array-val>
    </j-let>
    <j-let data="arrDisplay" data-eq data-let>
      <j-value-obr/></j-let>
     <j-let data="y = 0" data-let></j-let>
     <j-for start="let x" data-of="arrCollection" data-for>
        <j-set data="arrDisplay += x" data-set></j-set>
        <j-if data="y<3" data-if>
          <j-set data="arrDisplay" data-add-eq data-set>
            <j-value-obr/></j-set>
        </j-if>
        <j-set data="y++" data-set></j-set>
    </j-for>

   <j-append element="'#arrayTest'" data-append-block>
     <j-value data="arrDisplay" data-value></j-value>
   </j-append>
</jh-script>
```

- **Includes**
  - Check if the element array value is present on the set of array elements

```
<j-let data="included" data-eq data-let>
  <j-array-val data="fruits" method="array" type="includes"
    param="'Mango'" data-array-method></j-array-val>
</j-let>
```

```
<p style="margin:0;" id="arrayTest">
    <span style="color:red;">Fruit Included:</span>
  </p>
  <jh-script>
      <j-let data="fruits" data-eq data-let>
      <j-array>
          <j-arel data-dq="Banana" data-arel-dq></j-arel>
          <j-arel data-dq="Orange" data-arel-dq></j-arel>
          <j-arel data-dq="Apple" data-arel-dq></j-arel>
          <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
        </j-array>
      </j-let>
      <j-let data="included" data-eq data-let>
        <j-array-val data="fruits" method="array" type="includes"
          param="'Mango'" data-array-method></j-array-val>
      </j-let>

     <j-append element="'#arrayTest'" data-append-block>
       <j-value data="included" data-value></j-value>
     </j-append>
  </jh-script>
```

- **Extended Methods**
  - These are array methods that are exclusive on Jaihoe
    - **Is Array**
      - Check if the variable is an array

```
<j-array-val method="jaihoe" type="isArray"
  param="fruits" data-array-method></j-array-val>
```

```
<p style="margin:0;" id="arrTest"><b>Is Array?: </b></p>
<jh-script>
   <j-let data="fruits" data-eq data-let>
      <j-array>
        <j-arel data-dq="Banana" data-arel-dq></j-arel>
        <j-arel data-dq="Orange" data-arel-dq></j-arel>
        <j-arel data-dq="Apple" data-arel-dq></j-arel>
        <j-arel data-dq="Mango" data-arel-dq-close></j-arel>
      </j-array>
   </j-let>

      <j-append element="'#arrTest'" data-append-block>
        <j-array-val method="jaihoe" type="isArray"
          param="fruits" data-array-method></j-array-val>
      </j-append>
</jh-script>
```

    - **Unique Node Length**
      - Check if the length of the array element are same

```
<j-array-val method="jaihoe" type="uaLength"
  param="testA" data-array-method></j-array-val>
```

      - **For example:**
        - **"Nice"** length is **4** and **5000** length is also **4**, so it will return **true**
        - **"Apple"** length is **5** and **4200** length is **4**, so it will return **false**

- **Unique Node Length**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="arrTest"><b>Same Length?: </b></p>
<jh-script>
   <j-let data="testA" data-eq data-let>
    <j-array>
     <j-arel data-dq="Nice" data-arel-dq></j-arel>
     <j-arel data="5000" data-arel-close></j-arel>
    </j-array>
   </j-let>
   <j-let data="testB" data-eq data-let>
     <j-array>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data="4200" data-arel-close></j-arel>
     </j-array>
    </j-let>

  <j-append element="'#arrTest'" data-append-block>
    <j-value-rbr/>
    <j-array-val method="jaihoe" type="uaLength"
      param="testA" data-array-method></j-array-val><j-value-br/>
    <j-array-val method="jaihoe" type="uaLength"
      param="testB" data-array-method></j-array-val>
  </j-append>
</jh-script>
```

- **Unique Node Type**
  - Check if the type of the array element are the same

```
<j-array-val method="jaihoe" type="uaType"
  param="testA" data-array-method></j-array-val>
```

  - **For example**:
    - **"Nice"** is a **string** and **5000** is a **number**, so it will return **false**
    - **"Apple"** is a *string* and **"4200"** is number as a **string**, both are **string** so it will return **true**

- **Unique Node Type**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="arrTest"><b>Same Type?: </b></p>
<jh-script>

   <j-let data="testA" data-eq data-let>
    <j-array>
     <j-arel data-dq="Nice" data-arel-dq></j-arel>
     <j-arel data="5000" data-arel-close></j-arel>
    </j-array>
   </j-let>

   <j-let data="testB" data-eq data-let>
    <j-array>
     <j-arel data-dq="Apple" data-arel-dq></j-arel>
     <j-arel data-dq="4200" data-arel-dq-close></j-arel>
    </j-array>
   </j-let>

  <j-append element="'#arrTest'" data-append-block>
    <j-value-rbr/>
    <j-array-val method="jaihoe" type="uaType"
      param="testA" data-array-method></j-array-val><j-value-br/>

    <j-array-val method="jaihoe" type="uaType"
      param="testB" data-array-method></j-array-val>
  </j-append>

</jh-script>
```

- **Unique Node Values**
  - Check if the value of the array element are the same

  - **For example:**
    - **"Orange"** comparing to **"Orange"** is **equal**, so it will return **true**
    - **"Apple"** comparing to **5000** is not equal, so it will return **false**

- **Unique Node Values**
  - Here is the working code of the preview

```
<p style="margin:0;" id="arrTest"><b>Same Values?: </b></p>
<jh-script>
  <j-let data="testA" data-eq data-let>
   <j-array>
    <j-arel data-dq="Orange" data-arel-dq></j-arel>
    <j-arel data-dq="Orange" data-arel-dq-close></j-arel>
   </j-array>
  </j-let>
  <j-let data="testB" data-eq data-let>
   <j-array>
    <j-arel data-dq="Apple" data-arel-dq></j-arel>
    <j-arel data="5000" data-arel-close></j-arel>
   </j-array>
  </j-let>

 <j-append element="'#arrTest'" data-append-block>
   <j-value-rbr/>
   <j-array-val method="jaihoe" type="uaValues"
     param="testA" data-array-method></j-array-val><j-value-br/>
   <j-array-val method="jaihoe" type="uaValues"
     param="testB" data-array-method></j-array-val>
 </j-append>
</jh-script>
```

- **Clean Array**
  - Removes array elements with empty, whitespaces, null values

```
<j-array-val method="jaihoe" type="cleanArray"
  param="cleanArr" data-array-method></j-array-val>
```

  - **For example**:
    - **The Array Contains**:

```
["Apple",5,null,undefined,"      ",,,,"Orange"]
```

    - The **array element** to be **removed** are:

```
[null,undefined,"      ",,,]
```

    - The **array element** to be **returned** are:

```
["Apple",5,"Orange"]
```

- **Clean Array**
  - Here is the working example of the preview:

```
<p style="margin:0;" id="arrTest"><b>Array Set: </b></p>
<jh-script>
  <j-let data="cleanArr" data-eq data-let>
    <j-array>
      <j-arel data-dq="Apple" data-arel-dq></j-arel>
      <j-arel data="5" data-arel></j-arel>
      <j-arel data="null" data-arel></j-arel>
      <j-arel data="undefined" data-arel></j-arel>
      <j-arel data-dq="      " data-arel-dq></j-arel>
      <j-arel data="" data-arel></j-arel>
      <j-arel data="" data-arel></j-arel>
      <j-arel data="" data-arel></j-arel>
      <j-arel data-dq="Orange" data-arel-dq-close></j-arel>
    </j-array>
 </j-let>

  <j-append element="'#arrTest'" data-append-block>
    <j-array-val method="jaihoe" type="cleanArray"
      param="cleanArr" data-array-method></j-array-val>
    </j-append>
</jh-script>
```

- **Each Array Node**
  - Returns an object that counts the occurrence of the array node in an array.

```
<j-array-val method="jaihoe" type="eachArrNode"
  param="fruits" data-array-method></j-array-val>
```

- Here is a working code of using **Each Array Node** in Jaihoe:

```
<p style="word-wrap: break-word;" id="arrayTest">
 <span style="color:red;">Array:</span>
</p>
<jh-script>
  <j-let data="fruits" data-litarr data-let></j-let>
   <j-arrayx data="fruits" data-arrayx>
    <j-arelx param="'Banana',5" data-arelx></j-arelx>
    <j-arelx param="'Apple',1" data-arelx></j-arelx>
    <j-arelx param="'Orange','3'" data-arelx></j-arelx>
    <j-arelx param="'Mango'" data-arelx></j-arelx>
   </j-arrayx>
   <j-let data="fruitCount" data-eq data-let>
     <j-array-val method="jaihoe" type="eachArrNode"
       param="fruits" data-array-method></j-array-val>
   </j-let>
   <j-append element="'#arrayTest'" data-append-block>
     <j-stringify param="fruitCount" data-stringify></j-stringify>
   </j-append>
</jh-script>
```

- **Multiply Array Node (j-arrayx)**
  - Returns the current node multiple times.

```
<j-arrayx data="fruits" data-arrayx>
  <j-arelx param="query,queryLen" data-arelx></j-arelx>
  <j-arelx param="query,queryLen" data-arelx-close></j-arelx>
</j-arrayx>
```

```
<p style="word-wrap: break-word;" id="arrayTest">
    <span style="color:red;">Array:</span>
  </p>
  <jh-script>
     <j-let data="fruits" data-litarr data-let></j-let>
     <j-arrayx data="fruits" data-arrayx>
       <j-arelx param="'Banana',5" data-arelx></j-arelx>
       <j-arelx param="'Apple',1" data-arelx></j-arelx>
       <j-arelx param="'Orange','3'" data-arelx></j-arelx>
       <j-arelx param="'Mango'" data-arelx></j-arelx>
     </j-arrayx>
     <j-append element="'#arrayTest'" data-append-block>
       <j-value data="fruits" data-value />
     </j-append>
  </jh-script>
```

- **Multiply Array Node (j-arrayx)**
  - **Rules**:
    - Define an empty array for this method first
    - Node length can be in string but if its not a number it will return 1
    - Node length can be omitted or not included in the parameter
    - The third and succeeding parameter will be ignored so it will still be added to the array

```
<j-arelx param="'Pears',1,'f'" data-arelx-close></j-arelx>
```

    - Node length as 0 will not be added to the array

```
<j-arelx param="'Pears',0" data-arelx-close></j-arelx>
```

    - Multiple Array Node only accepts Arrays, not strings or numbers so it will not be added to the array

```
<j-arel data-dq="Watermelon" data-arel-dq></j-arel>
```

- **Sets**
  - Similar to array, but values must be unique
    - **New Set**
      - For creating a new set

```
<j-nset></j-nset>
```

```
<p style="margin:0;" id="setTest"><b>New Set: </b></p>
<jh-script>
  <j-let data="letters" data-eq data-let>
    <j-nset><j-array>
        <j-arel data-dq="a" data-arel-dq></j-arel>
      <j-arel data-sq="b" data-arel-sq></j-arel>
      <j-arel data="3" data-arel-close></j-arel>
    </j-array></j-nset></j-let>
 <j-append element="'#setTest'" data-append-block>
  <j-value data="Array.from(letters)" data-value></j-value>
 </j-append>
</jh-script>
```

- **New Set**
  - For creating a **new set**:
    - Put in a variable block like **<j-var>** or **<j-let>**
    - Then inside make a **<j-nset>** new set block for **new Set()**,
    - then **<j-array>** array block for **[ ]** holding the values,
    - then **<j-arel>** for the array elements, visit array for array element implementation.

  - **Methods**
    - **Add**
      - A method that adds a new element to the set, using the **<j-sblock> (set block method)**

```
<j-sblock data="letters" method="set" type="add" param="'d'"
  data-sblock-method></j-sblock>
```

```
<p style="margin:0;" id="setTest"><b>New Set: </b></p>
  <jh-script>
    <j-let data="letters" data-eq data-let>
      <j-nset><j-array>
          <j-arel data-dq="a" data-arel-dq></j-arel>
        <j-arel data-sq="b" data-arel-sq></j-arel>
        <j-arel data="3" data-arel-close></j-arel>
      </j-array></j-nset></j-let>

  <j-sblock data="letters" method="set" type="add" param="'c'"
    data-sblock-method></j-sblock>

   <j-append element="'#setTest'" data-append-block>
    <j-value data="Array.from(letters)" data-value></j-value>
   </j-append>
  </jh-script>
```

- **Delete**
  - A method that remove a part of the element in the set, using the **<j-sblock> (set block method)**

```
<j-sblock data="letters" method="set" type="delete" param="3"
  data-sblock-method></j-sblock>
```

```
<p style="margin:0;" id="setTest"><b>New Set: </b></p>
  <jh-script>
    <j-let data="letters" data-eq data-let>
      <j-nset><j-array>
          <j-arel data-dq="a" data-arel-dq></j-arel>
        <j-arel data-sq="b" data-arel-sq></j-arel>
        <j-arel data="3" data-arel-close></j-arel>
      </j-array></j-nset></j-let>

  <j-sblock data="letters" method="set" type="add" param="'c'"
    data-sblock-method></j-sblock>
  <j-sblock data="letters" method="set" type="delete" param="3"
    data-sblock-method></j-sblock>

   <j-append element="'#setTest'" data-append-block>
    <j-value data="Array.from(letters)" data-value></j-value>
   </j-append>
  </jh-script>
```

- **Has**
  - Returns the condition if the value exist within the set, using the **<j-svalue> (set value method)**

```
<j-svalue data="letters" method="set" type="has" param="'b'"
    data-svalue-method></j-svalue>
```

- **Has**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="setTest"><b>Set Has?: </b></p>
  <jh-script>
    <j-let data="letters" data-eq data-let>
      <j-nset><j-array>
          <j-arel data-dq="a" data-arel-dq></j-arel>
        <j-arel data-sq="b" data-arel-sq></j-arel>
      </j-array></j-nset></j-let>

   <j-append element="'#setTest'" data-append-block>
    <j-value-rbr/>
    <j-svalue data="letters" method="set" type="has" param="'b'"
      data-svalue-method></j-svalue><j-value-br/>
    <j-svalue data="letters" method="set" type="has" param="'c'"
      data-svalue-method></j-svalue>
   </j-append>
  </jh-script>
```

- **For Each**
  - Method that calls each item of the set
  - using the following blocks
    - **<j-sblock> (set block method)**
    - **<j-fx> (regular function block)**

```
<j-sblock data="letters" method="set" type="forEach" data-sblock-fx>
  <j-fx param="value" data-fx>
    <!--INSERT CODE HERE -->
  </j-fx>
</j-sblock>
```

- **For Each**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="setTest"><b>ForEach Set: </b></p>
<jh-script>
    <j-let data="letters" data-eq data-let>
      <j-nset><j-array>
        <j-arel data-dq="a" data-arel-dq></j-arel>
        <j-arel data-sq="b" data-arel-sq></j-arel>
        <j-arel data-sq="c" data-arel-sq></j-arel>
      </j-array></j-nset></j-let>

 <j-let data="text = '" data-let></j-let>
 <j-sblock data="letters" method="set" type="forEach" data-sblock-fx>
   <j-fx param="value" data-fx>
     <j-set data="text += value" data-add data-br data-set></j-set>
   </j-fx>
 </j-sblock>

   <j-append element="'#setTest'" data-append-block>
    <j-value-rbr/><j-value data="text" data-value></j-value>
   </j-append>
</jh-script>
```

- **Values**
  - Returns the iterator object of with its values of the set

```
<j-let data="text = '" data-let></j-let>
 <j-for start="x" data-of="letters" data-set-values data-for>
   <j-set data="text += x" data-add data-br data-set></j-set>
 </j-for>
```

  - Using **for of** loop with **data-set-values** (which represents **.values()**)
  - Check **Loops** for more information about types of for loops

- **Values**
  - Here is the working code of the preview

```
<p style="margin:0;" id="setTest"><b>Set Values: </b></p>
 <jh-script>
   <j-let data="letters" data-eq data-let>
     <j-nset><j-array>
       <j-arel data-dq="a" data-arel-dq></j-arel>
       <j-arel data-sq="b" data-arel-sq></j-arel>
       <j-arel data-sq="c" data-arel-sq></j-arel>
     </j-array></j-nset></j-let>

 <j-let data="text = '''" data-let></j-let>
  <j-for start="x" data-of="letters" data-set-values data-for>
   <j-set data="text += x" data-add data-br data-set></j-set>
  </j-for>

  <j-append element="'#setTest'" data-append-block>
   <j-value-rbr/><j-value data="text" data-value></j-value>
  </j-append>
 </jh-script>
```

- **Property**
  - **Size**
    - Literally returns the size of the set

```
<j-svalue type="property" data="letters" data-set-size>
</j-svalue>
```

    - This block **uses**:
      - **<j-svalue> set value method (block)** including:
      - **type="property"** to access its property
      - **data="variable"** to target the set variable
      - **data-set-size** to get the size of the set
    - Since it uses property, not method they have a different block structure

- **Size**
  - Here is the working code of the preview

```
<p style="margin:0;" id="setTest"><b>Set Size: </b></p>
  <jh-script>
    <j-let data="letters" data-eq data-let>
      <j-nset><j-array>
          <j-arel data-dq="a" data-arel-dq></j-arel>
        <j-arel data-sq="b" data-arel-sq></j-arel>
        <j-arel data-sq="c" data-arel-sq-close></j-arel>
      </j-array></j-nset></j-let>

   <j-append element="'#setTest'" data-append-block>
    <j-svalue type="property" data="letters" data-set-size>
     </j-svalue>
   </j-append>
  </jh-script>
```

- **Maps**
  - **Methods**
    - **New Map**
      - Literally creates a new map, and there are **2 declarative implementations** and **1 method implementation** (which is set) in creating a **new map**
      - **Declarative Implementation**
        - **Direct Implementation**
          - Creating a new map requires **<j-arset> Array Set** and this is how to implement it

```
<j-nmap><j-array>
  <j-arset data="'apples',500" data-arset></j-arset>
  <j-arset data="'bananas',300" data-arset></j-arset>
  <j-arset data="'oranges',200"
    data-arset-close></j-arset>
</j-array></j-nmap>
```

          - There is no double or single quote implementation here because its an array set
          - If it will be the last set of the map, you have to close it with data-arset-close

        - **Nested Implementation**
          - Creating new a new map requires **<j-array-add>** for the first and succeeding array set and **<j-array>** as the last array set
          - To implement it, see in the next page

- **Nested Implementation**
  - Here is how you **implement** it:
  - In here you can now use **single** (data-sq) and **double** (data-dq) quote implementation because array elements are separated

```
<j-nmap><j-array>
  <j-array-add>
   <j-arel data-dq="apples" data-arel-dq></j-arel>
   <j-arel data="500" data-arel-close></j-arel>
  </j-array-add>
  <j-array-add>
   <j-arel data-dq="bananas" data-arel-dq></j-arel>
   <j-arel data="300" data-arel-close></j-arel>
  </j-array-add>
  <j-array>
   <j-arel data-dq="oranges" data-arel-dq></j-arel>
   <j-arel data="200" data-arel-close></j-arel>
  </j-array>
</j-array></j-nmap>
```

- **All together**: This is the working code **(New Map)**

```
<p style="margin:0;" id="mapTest"><b>NEW MAP: </b></p>
  <jh-script>
    <j-let data="fruitPriceA" data-eq data-let>
      <j-nmap><j-array>
          <j-arset data="'apples',500" data-arset></j-arset>
          <j-arset data="'bananas',300" data-arset></j-arset>
          <j-arset data="'oranges',200" data-arset-close></j-arset>
      </j-array></j-nmap></j-let>

     <j-let data="fruitPriceB" data-eq data-let>
      <j-nmap><j-array>
        <j-array-add><j-arel data-dq="apples" data-arel-dq></j-arel>
        <j-arel data="500" data-arel-close></j-arel></j-array-add>
        <j-array-add><j-arel data-dq="bananas" data-arel-dq></j-arel>
        <j-arel data="300" data-arel-close></j-arel></j-array-add>
        <j-array><j-arel data-dq="oranges" data-arel-dq></j-arel>
        <j-arel data="200" data-arel-close></j-arel></j-array>
      </j-array></j-nmap></j-let>

   <j-append element="'#mapTest'" data-append-block>
     <j-value-rbr/>
     <j-value data="Array.from(fruitPriceA)" data-value></j-value>
     <j-value-br/>
     <j-value data="Array.from(fruitPriceB)" data-value></j-value>
   </j-append>
</jh-script>
```

- **All together:** This is the preview **(New Map)**

  - **As you can see, it outputs the same value**

- **Methods**
  - **Set**
    - A method that adds a key with its value to the map, using the **<j-mblock> map block method**

```
<j-mblock data="fruitSet" method="map" type="set"
  param="'apple',500" data-mblock-method></j-mblock>
```

```
<p style="margin:0;" id="mapTest"><b>NEW MAP: </b></p>
<jh-script>
  <j-let data="fruitSet" data-eq data-let>
    <j-nmap></j-nmap>
  </j-let>

    <j-mblock data="fruitSet" method="map" type="set"
      param="'apple',500" data-mblock-method></j-mblock>
    <j-mblock data="fruitSet" method="map" type="set"
      param="'bananas',300" data-mblock-method></j-mblock>
     <j-mblock data="fruitSet" method="map" type="set"
       param="'oranges',200" data-mblock-method></j-mblock>

  <j-append element="'#mapTest'" data-append-block>
    <j-value-rbr/>
    <j-value data="Array.from(fruitSet)" data-value></j-value>
  </j-append>
</jh-script>
```

- **Get**
  - A method the returns the value of th key from the map, using the **<j-mvalue> map value method**

```
<j-mvalue data="fruitSet" method="map" type="get"
  param="'apples'" data-mvalue-method></j-mvalue>
```

```
<p style="margin:0;" id="mapTest"><b>MAP VALUES: </b></p>
<jh-script>
   <j-let data="fruitSet" data-eq data-let>
    <j-nmap><j-array>
     <j-arset data="'apples',500" data-arset></j-arset>
     <j-arset data="'bananas',300" data-arset></j-arset>
     <j-arset data="'oranges',200" data-arset-close></j-arset>
   </j-array></j-nmap></j-let>

   <j-append element="'#mapTest'" data-append-block>
     <j-value-rbr/>
     <j-mvalue data="fruitSet" method="map" type="get"
       param="'apples'" data-mvalue-method></j-mvalue>
      <j-value-br/>
     <j-mvalue data="fruitSet" method="map" type="get"
       param="'bananas'" data-mvalue-method></j-mvalue>
      <j-value-br/>
     <j-mvalue data="fruitSet" method="map" type="get"
       param="'oranges'" data-mvalue-method></j-mvalue>
   </j-append>
</jh-script>
```

  - As you can see:
    - **<j-mblock>** map block method is for dependent method implementation
    - **<j-mvalue>** map value method is for a valued oriented method for variables, append or write elements and etc.

- **Delete**
  - A method that removes a key with its value (map element) from the map, using the **<j-mblock> map block method**

```
<j-mblock data="fruitSet" method="map" type="delete"
  param="'apples'" data-mblock-method></j-mblock>
```

```
<p style="margin:0;" id="mapTest"><b>MAP SET: </b></p>
<jh-script>
   <j-let data="fruitSet" data-eq data-let>
     <j-nmap><j-array>
       <j-arset data="'apples',500" data-arset></j-arset>
       <j-arset data="'bananas',300" data-arset></j-arset>
       <j-arset data="'oranges',200" data-arset-close></j-arset>
     </j-array></j-nmap></j-let>

   <j-mblock data="fruitSet" method="map" type="delete"
     param="'apples'" data-mblock-method></j-mblock>

   <j-append element="'#mapTest'" data-append-block>
     <j-value-rbr/>
     <j-value data="Array.from(fruitSet)" data-value></j-value>
   </j-append>
</jh-script>
```

- **Has**
  - A method that checks if the key exist in the map

```
<j-mvalue data="fruitSet" method="map" type="has"
  param="'apples'" data-mvalue-method></j-mvalue>
```

- **Has**
  - Here is the working code of the preview

```
<p style="margin:0;" id="mapTest"><b>MAP HAS: </b></p>
<jh-script>
   <j-let data="fruitSet" data-eq data-let>
    <j-nmap><j-array>
     <j-arset data="'apples',500" data-arset></j-arset>
     <j-arset data="'bananas',300" data-arset-close></j-arset>
   </j-array></j-nmap></j-let>

   <j-append element="'#mapTest'" data-append-block>
     <j-value-rbr/>
     <j-mvalue data="fruitSet" method="map" type="has"
       param="'apples'" data-mvalue-method></j-mvalue>
      <j-value-br/>
     <j-mvalue data="fruitSet" method="map" type="has"
       param="'oranges'" data-mvalue-method></j-mvalue>
   </j-append>
</jh-script>
```

- **For Each**
  - Method that calls each key with its values of the map
  - Using the following blocks:
    - **<m-block> (map block method)**
    - **<j-mblock-fx> (special map function block)**

```
<j-mblock data="fruitSet" method="map" type="forEach"
  data-mblock-fx>
   <j-mblock-fx param="value, key" data-fx>
     <!--INSERT CODE HERE -->
   </j-mblock-fx>
</j-mblock>
```

- **For Each**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="mapTest"><b>MAP FOR-EACH: </b></p>
<jh-script>
   <j-let data="fruitSet" data-eq data-let>
    <j-nmap><j-array>
     <j-arset data="'apples',500" data-arset></j-arset>
     <j-arset data="'bananas',300" data-arset></j-arset>
     <j-arset data="'oranges',200" data-arset-close></j-arset>
   </j-array></j-nmap></j-let>

   <j-let data="text = '" data-let></j-let>
   <j-mblock data="fruitSet" method="map" type="forEach"
     data-mblock-fx>
      <j-mblock-fx param="value, key" data-fx>
        <j-set data="text += key + ' = ' + value +" data-br
          data-set></j-set>
      </j-mblock-fx>
    </j-mblock>

   <j-append element="'#mapTest'" data-append-block>
     <j-value-rbr/>
     <j-value data="text" data-value></j-value>
   </j-append>
</jh-script>
```

- **(For of) Methods**
  - **Entries**
    - Returns the [key, values] from the map, using the data-map-entries attribute

```
<j-for start="let x" data-of="fruitSet" data-map-entries
  data-for>
</j-for>
```

- **Entries**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="mapTest"><b>MAP FOR-OF ENTRIES: </b></p>
<jh-script>
   <j-let data="fruitSet" data-eq data-let>
    <j-nmap><j-array>
     <j-arset data="'apples',500" data-arset></j-arset>
     <j-arset data="'bananas',300" data-arset></j-arset>
     <j-arset data="'oranges',200" data-arset-close></j-arset>
   </j-array></j-nmap></j-let>

   <j-let data="text = ''" data-let></j-let>
   <j-for start="let x" data-of="fruitSet" data-map-entries
     data-for>
     <j-set data="text += x" data-add data-br data-set></j-set>
   </j-for>

   <j-append element="'#mapTest'" data-append-block>
     <j-value-rbr/>
     <j-value data="text" data-value></j-value>
   </j-append>
</jh-script>
```

- **Keys**
  - Returns each key from the map, using the data-map-keys attribute

```
<j-for start="let x" data-of="fruitSet" data-map-keys data-for>
</j-for>
```

  - As you could **see**:
    - Apples
    - Bananas
    - Oranges

  - The **keys** are only listed!

- **Keys**

- Here is the working code of the preview:

```
<p style="margin:0;" id="mapTest"><b>MAP FOR-OF KEYS: </b></p>
<jh-script>
   <j-let data="fruitSet" data-eq data-let>
    <j-nmap><j-array>
     <j-arset data="'apples',500" data-arset></j-arset>
     <j-arset data="'bananas',300" data-arset></j-arset>
     <j-arset data="'oranges',200" data-arset-close></j-arset>
   </j-array></j-nmap></j-let>

   <j-let data="text = ''" data-let></j-let>
  <j-for start="let x" data-of="fruitSet" data-map-keys data-for>
    <j-set data="text += x" data-add data-br data-set></j-set>
  </j-for>

   <j-append element="'#mapTest'" data-append-block>
     <j-value-rbr/>
     <j-value data="text" data-value></j-value>
   </j-append>
</jh-script>
```

- **Values**
  - Returns each value from the map, using the data-map-values attribute

```
<j-for start="let x" data-of="fruitSet" data-map-values data-for>

</j-for>
```

  - As you could **see**:
    - 500
    - 300
    - 200

  - The **values** are only listed!

- **Values**

- Here is the working code of the preview:

```
<p style="margin:0;" id="mapTest"><b>MAP FOR-OF VALUES: </b></p>
<jh-script>
   <j-let data="fruitSet" data-eq data-let>
    <j-nmap><j-array>
     <j-arset data="'apples',500" data-arset></j-arset>
     <j-arset data="'bananas',300" data-arset></j-arset>
     <j-arset data="'oranges',200" data-arset-close></j-arset>
   </j-array></j-nmap></j-let>

   <j-let data="text = ''" data-let></j-let>
   <j-for start="let x" data-of="fruitSet" data-map-values
     data-for>
     <j-set data="text += x" data-add data-br data-set></j-set>
   </j-for>

   <j-append element="'#mapTest'" data-append-block>
     <j-value-rbr/>
     <j-value data="text" data-value></j-value>
   </j-append>
</jh-script>
```

- **Property**
  - **Size**
    - Returns the size of the map

```
<j-mvalue data="fruitSet" data-map-size data-mvalue></j-mvalue>
```

```
<p style="margin:0;" id="mapTest"><b>MAP SIZE: </b></p>
<jh-script>
   <j-let data="fruitSet" data-eq data-let>
    <j-nmap><j-array>
     <j-arset data="'apples',500" data-arset></j-arset>
     <j-arset data="'bananas',300" data-arset></j-arset>
     <j-arset data="'oranges',200" data-arset-close></j-arset>
   </j-array></j-nmap></j-let>

 <j-append element="'#mapTest'" data-append-block>
  <j-mvalue data="fruitSet" data-map-size data-mvalue></j-mvalue>
 </j-append>
</jh-script>
```

- **Classes**
  - A block template for making objects, but its not literally an object but a collection of functions as class objects
    - **Foundation (j-class)**
      - This is how you create a class block

```
<j-class name="className" data-class>
</j-class>
```

    - **Constructor (j-class-cfx)**
      - In **Jaihoe** this is called **constructor function <j-class-cfx>** and this block must always inside a class block except the parameter its up to you

```
<j-class-cfx param="name, year" data-class-cfx>
  <j-set data-this="name" data-eq="name" data-set></j-set>
  <j-set data-this="year" data-eq="year" data-set></j-set>
</j-class-cfx>
```

    - **Class Function (j-class-fx)**
      - These are additional **function methods** that you can add to the class block

```
<j-class-fx name="age" param="age" data-class-fx>
  <j-let data="date" data-eq="new Date()" data-let></j-let>
  <j-return value="date.getFullYear()"
    data-minus data-this="year" data-return></j-return>
</j-class-fx>
```

- **All together (Preview)**
  - This is an example on how you implement a class in JaihoeScript form

- **All together (Code)**
  - Here is the working code of the preview:

```
<p style="margin:0;" id="mapTest"><b>CAR CLASS INFO: </b></p>
<jh-script>

  <j-class name="Car" data-class>
    <j-class-cfx param="name, year" data-class-cfx>
      <j-set data-this="name" data-eq="name" data-set></j-set>
      <j-set data-this="year" data-eq="year" data-set></j-set>
    </j-class-cfx>

    <j-class-fx name="age" param data-class-fx>
      <j-let data="date" data-eq="new Date()" data-let></j-let>
      <j-return value="date.getFullYear()" data-minus
        data-this="year" data-return></j-return>
    </j-class-fx>
  </j-class>

   <j-let data="myCar" data-eq data-new="Car"
     param="'Ford',2014" data-let-param></j-let>
   <j-append element="'#mapTest'" data-append-block>
     <j-value-rbr/>
      <j-value data="'My car name is '+myCar.name"
        data-value></j-value>
      <j-value-br/>
     <j-value data="'My car age is '+myCar.age"
       data-value-cparam></j-value>
   </j-append>

</jh-script>
```

  - For the **attributes**:
    - data-this is equal to “this.”
    - data-eq=”value” is equal to “= value”
    - data-new=”Class” is equal to “new Class” for adding a new class
    - param=”items” is equal to “(items” for adding parameters
    - data-let-param is equal to “)” for closing parameters

- **TypeOf**
  - **Foundation**
    - Returns the type of the variable

```
<j-typeof data="3.14" data-typeof></j-typeof>
```

      - This variable is stated as a number, so it will return “number”

- **All together (Preview and Code)**
  - Here is the preview of using TypeOf

  - Here is the **working code** of the preview:

```
<p style="margin:0;" id="mapTest"><b>TYPEOF CATEGORIES: </b></p>
<jh-script>
   <j-append element="'#mapTest'" data-append-block>
     <j-value-rbr/>
     <j-value data="`'John' is `" data-add data-value></j-value>
     <j-typeof data="'John'" data-typeof></j-typeof><j-value-br/>
     <j-value data="'3.14 is '" data-add data-value></j-value>
     <j-typeof data="3.14" data-typeof></j-typeof><j-value-br/>
     <j-value data="'NaN is '" data-add data-value></j-value>
     <j-typeof data="NaN" data-typeof></j-typeof><j-value-br/>
     <j-value data="'false is '" data-add data-value></j-value>
     <j-typeof data="false" data-typeof></j-typeof><j-value-br/>
     <j-value data="'[1,2,3,4] is '" data-add data-value></j-value>
     <j-typeof data="[1,2,3,4]" data-typeof></j-typeof><j-value-br/>
     <j-value data="`{name:'James',age:25} is `" data-add data-value></j-value>
     <j-typeof data="{name:'James',age:25}" data-typeof></j-typeof><j-value-br/>
     <j-value data="'new Date() is '" data-add data-value></j-value>
     <j-typeof data="new Date()" data-typeof></j-typeof><j-value-br/>
     <j-value data="'function(){} is '" data-add data-value></j-value>
     <j-typeof data="function(){}" data-typeof></j-typeof><j-value-br/>
     <j-value data="'myCar is '" data-add data-value></j-value>
     <j-typeof data="myCar" data-typeof></j-typeof><j-value-br/>
     <j-value data="'null is '" data-add data-value></j-value>
     <j-typeof data="null" data-typeof></j-typeof>
   </j-append>
</jh-script>
```

- **Time Methods**
  - Everything about time methods that are useful depending on the code purpose.
    - **Set Timeout**
      - Executes a block of code with a delay

```
<j-set-time>
   <j-fx param data-fx>
      <!--ENTER YOUR BLOCK OF CODE HERE-->
   </j-fx>
   <j-timeout time="3000"></j-timeout>
</j-set-time>
```

```
<p id="status" style="margin:0;display:inline;">
  Do you want to order a food?
</p>
<button id="order" type="button">Order Food</button>
<jh-script>
  <j-add-evt element="#order" data-elem-evtype="click"
    data-add-evt>
  <j-fx param data-fx>
    <j-write element="'#status'" method="write"
      output="'Food order is coming in 3 seconds'"
      data-write></j-write>
    <j-set-time>
      <j-fx param data-fx>
        <j-write element="'#status'" method="write"
          output="'Order arrived, eat now! Then order again.'"
          data-write></j-write>
      </j-fx>
      <j-timeout time="3000"></j-timeout>
   </j-set-time>
  </j-fx>
  </j-add-evt>
</jh-script>
```

      - Alternatively you can use set timeout in **one line with function**

```
<j-set-time data-fx="updateStatus" data-timeout="3000"
  data-set-time></j-set-time>
```

        - If you have to **define** this set time, use the **set block method**

```
<j-let data="order" data-let></j-let>
<j-set-block data="order" data-eq data-set>
  <j-set-time data-fx="updateStatus"
    data-timeout="3000" data-set-time></j-set-time>
</j-set-block>
```

- **Set Timeout**
  - Here is the working code of the **one line with a function**

```
<p id="status" style="margin:0;display:inline;">
  Your package is coming in 5 seconds.
</p>
<jh-script>
   <j-set-time data-fx="updateStatus" data-timeout="5000"
     data-set-time></j-set-time>
   <j-fx name="updateStatus" param data-fx>
     <j-write element="'#status'" method="write"
       output="'Order arrived, please check it now.'"
       data-write></j-write>
   </j-fx>
</jh-script>
```

- **Clear Timeout**
  - Literally stops a set timeout function if defined

```
<j-clear-time data="order" data-clear-time></j-clear-time>
```

  - Two case **scenarios**:
    - If you click **cancel order**, it will cancel the time arrival of the order, try to wait for 5 seconds it will not update because of clear timeout
    - If you click **check**, it will update the status to please wait, try to wait for 5 seconds it will still update because clear timeout is not defined but if you press it on the cancel order state it will not update

- **Clear Timeout**
  - Here is the working code of the preview:

```
<p id="status" style="margin:0;display:inline;"></p>
<button id="cancel" type="button">Cancel Order</button>
<button id="orderItem" type="button" hidden>Order</button>
<button id="check" type="button">Check</button>

<jh-script>

 <j-let data="order" data-let></j-let>
 <j-fx name="orderPackage" param data-fx>
   <j-write element="'#status'" method="write"
     output="'Your package is coming in 5 seconds.'" data-write></j-write>
   <j-set-block data="order" data-eq data-set>
    <j-set-time data-fx="updateStatus"
      data-timeout="5000" data-set-time></j-set-time>
   </j-set-block>
 </j-fx>
 <j-fx-call name="orderPackage" param data-fx-call></j-fx-call>
 <j-fx name="updateStatus" param data-fx>
   <j-write element="'#status'" method="write"
     output="'Order arrived, again?'" data-write></j-write>
   <j-docget data-id="'orderItem'" data-prop="style.display"
     data-eq="'inline'" data-docget-apply></j-docget>
   <j-docget data-id="'cancel'" data-prop="style.display"
     data-eq="'none'" data-docget-apply></j-docget>
 </j-fx>
 <j-add-evt element="#orderItem" data-elem-evtype="click" data-add-evt>
   <j-fx param data-fx>
     <j-fx-call name="orderPackage" param data-fx-call></j-fx-call>
     <j-docget data-id="'orderItem'" data-prop="style.display"
       data-eq="'none'" data-docget-apply></j-docget>
     <j-docget data-id="'cancel'" data-prop="style.display"
       data-eq="'inline'" data-docget-apply></j-docget>
   </j-fx>
  </j-add-evt>
  <j-add-evt element="#cancel" data-elem-evtype="click" data-add-evt>
    <j-fx param data-fx>
      <j-write element="'#status'" method="write"
        output="'Order canceled, order again?'" data-write></j-write>
      <j-docget data-id="'orderItem'" data-prop="style.display"
        data-eq="'inline'" data-docget-apply></j-docget>
      <j-docget data-id="'cancel'" data-prop="style.display"
        data-eq="'none'" data-docget-apply></j-docget>
      <j-clear-time data="order" data-clear-time></j-clear-time>
    </j-fx>
   </j-add-evt>
  <j-add-evt element="#check" data-elem-evtype="click" data-add-evt>
   <j-fx param data-fx>
     <j-write element="'#status'" method="write"
      output="'Please wait or press the other buttons.'" data-write></j-write>
    </j-fx>
  </j-add-evt>
</jh-script>
```

❖ Set Interval

➢ Executes a block of code repeatedly with delay

```
<j-set-interval>
   <j-fx param data-fx>
      <!--INSERT CODE HERE-->
   </j-fx>
   <j-interval time="1000"></j-interval>
</j-set-interval>
```

```
<p style="margin:0;"><span>Lets count from <b>1 to 10</b>:</span>
<span id="number"></span></p>
<jh-script>
  <j-let data="num = 1" data-let></j-let>
  <j-set-interval>
   <j-fx param data-fx>
     <j-write element="'#number'" method="write" output="num"
       data-write></j-write>
     <j-set data="num" data-inc data-set></j-set>
     <j-if data="num > 10" data-if>
      <j-set data="num = 1" data-set></j-set>
     </j-if>
   </j-fx>
   <j-interval time="1000"></j-interval>
  </j-set-interval>
</jh-script>
```

➢ Alternatively you can use set interval in **one line with function**

```
<j-set-interval data-fx="countNum" data-interval="1000"
  data-set-interval></j-set-interval>
```

➢ If you have to **define** this set interval, use the **set block method**

```
<j-let data="countdown" data-let></j-let>
<j-set-block data="countdown" data-eq data-set>
  <j-set-interval data-fx="countNum" data-interval="1000"
    data-set-interval></j-set-interval>
</j-set-block>
```

❖ Set Interval

➢ Here is the working code of the preview

```
<p style="margin:0;"><span>Lets countdown from <b>(10)</b>:</span>
<span id="number"></span></p>

<jh-script>
  <j-let data="num = 10" data-let></j-let>
  <j-set-interval data-fx="countNum" data-interval="1000"
    data-set-interval></j-set-interval>

  <j-fx name="countNum" param data-fx>
    <j-write element="'#number'" method="write" output="num"
      data-write></j-write>
    <j-set data="num" data-dec data-set></j-set>
    <j-if data="num < 1" data-if>
      <j-set data="num = 10" data-set></j-set>
    </j-if>
  </j-fx>
</jh-script>
```

❖ Clear Interval

➢ Literally stops a set interval function if defined

```
<j-clear-interval data="countdown" data-clear-interval>
  </j-clear-interval>
```

➢ You can wrap it with an event listener

```
<j-add-evt element="#stopCountDown" data-elem-evtype="click"
  data-add-evt>
  <j-fx param data-fx>
    <j-clear-interval data="countdown" data-clear-interval>
     </j-clear-interval>
  </j-fx>
</j-add-evt>
```

➢ In that way it can be stop by the button.

- **Clear Interval**
  - Here is the working code of the preview:

```
<p style="margin:0;"><span>Lets countdown from <b>(10)</b>:</span>
<span id="number"></span></p>
<button id="stopCountDown" type="button">Stop Countdown</button>
<jh-script>
  <j-let data="num = 10" data-let></j-let>

  <j-let data="countdown" data-let></j-let>
  <j-set-block data="countdown" data-eq data-set-block>
    <j-set-interval data-fx="countNum" data-interval="1000"
      data-set-interval></j-set-interval>
  </j-set-block>

  <j-fx name="countNum" param data-fx>
    <j-write element="'#number'" method="write" output="num"
      data-write></j-write>
    <j-set data="num" data-dec data-set></j-set>
      <j-if data="num < 1" data-if>
        <j-set data="num = 10" data-set></j-set>
      </j-if>
  </j-fx>

  <j-add-evt element="#stopCountDown" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
      <j-clear-interval data="countdown" data-clear-interval>
        </j-clear-interval>
    </j-fx>
  </j-add-evt>
</jh-script>
```

- **Promise**
  - **Foundation**
    - Getting values may take some time depending on the situation like fetching data that times time to arrive.

```
<j-promise><j-fx param="resolve, reject" data-fx>
   <j-if data="returnBook == true" data-if>
     <j-fx-call name="resolve" param="'Returned'"
       data-fx-call></j-fx-call>
   </j-if>
   <j-else>
     <j-fx-call name="reject" param="'Not Returned'"
       data-fx-call></j-fx-call>
   </j-else>
</j-fx></j-promise>
```

- **Promise**
  - **Working Example**
    - Here is the preview and working code of a working promise

```
<p style="margin:0;"><span>Returning your <b>book</b>:</span>
<span id="status">Pending</span></p>

<jh-script>
    <!-- You can change the returnBook value to false -->
    <j-let data="returnBook = true" data-let></j-let>

    <j-let data="promiseToReturnBook" data-eq data-let>
    <j-promise><j-fx param="resolve, reject" data-fx>
      <j-set-time><j-fx param data-fx>
      <j-if data="returnBook == true" data-if>
        <j-fx-call name="resolve" param="'Returned'"
          data-fx-call></j-fx-call>
      </j-if>
      <j-else>
        <j-fx-call name="reject" param="'Not Returned'"
          data-fx-call></j-fx-call>
      </j-else>
     </j-fx><j-timeout time="3000"></j-timeout></j-set-time>
    </j-fx></j-promise></j-let>

    <j-then-close data="promiseToReturnBook" data-then>
    <j-fx param="value" data-fx>
      <j-write element="'#status'" method="write"
        output="value" data-write></j-write>
    </j-fx>
    <j-and/>
    <j-fx param="err" data-fx>
      <j-write element="'#status'" method="write"
        output="err" data-write></j-write>
    </j-fx>
    </j-then-close>

</jh-script>
```

- **Async and Await**
  - Aside from a regular function, **Async** are mostly functions that runs asynchronously, usually inside of the function **Await** is used as it waits a method like promise or fetch.
  - **Foundation**
    - This is how you use **async** in two ways
      - **Async Function**

```
<j-afx name="afxName" param data-afx>
  <!--INSERT CODE HERE-->
</j-afx>
```

      - **Async Appended Method**

```
<j-async/><j-fx name="convertFxToAFx" param data-fx>
  <!--INSERT CODE HERE-->
</j-fx>
```

    - This is how you use an **await** in three ways
      - **Await Variable Method (data-eq-await & data-as-wait)**

```
<!-- SHORT-HAND (= await is data-eq-await) -->
<j-let data="promiseDisplay" data-eq-await="promiseVar"
  data-let></j-let>
<j-write element="'#status'" method="write"
  output="promiseDisplay" data-write></j-write>
<!-- LONG-HAND (await is data-as-await) -->
<j-let data="promiseDisplay" data-eq data-as-await="promiseVar"
  data-let></j-let>
<j-write element="'#status'" method="write"
  output="promiseDisplay" data-write></j-write>
```

      - **Await Value Method (data-await)**

```
<j-write element="'#status'" data-write-block>
  <j-value data-await="promiseVar" data-value></j-value>
</j-write>
```

      - **Await Block Method <j-await ..></j-await>**

```
<j-write element="'#status'" data-write-block>
  <j-await data="promiseVar" data-await></j-await>
</j-write>
```

      - **Await Appended Method <j-await/>**

```
<j-write element="'#status'" data-write-block>
  <j-await/><j-value data="promiseVar" data-value></j-value>
</j-write>
```

- **Async and Await**
  - **Working Example**
    - Here is the preview and working code of using async and await

```
<p style="margin:0;"><span>Returning your <b>instrument</b>:</span>
<span id="status">Pending</span></p>

<jh-script>
  <j-afx name="returnInstrument" param data-afx>
     <!-- You can change the returnInst value to false -->
     <j-let data="returnInst = true" data-let></j-let>

        <j-let data="promiseToReturnInst" data-eq data-let>
        <j-promise><j-fx param="resolve, reject" data-fx>
        <j-set-time><j-fx param data-fx>
          <j-if data="returnInst == true" data-if>
            <j-fx-call name="resolve" param="'Returned'"
              data-fx-call></j-fx-call>
          </j-if>
          <j-else>
            <j-fx-call name="reject" param="'Not Returned'"
              data-fx-call></j-fx-call>
          </j-else>
        </j-fx><j-timeout time="3000"></j-timeout></j-set-time>
      </j-fx></j-promise></j-let>

      <j-try>
       <j-write element="'#status'" data-write-block>
         <j-await data="promiseToReturnInst" data-await></j-await>
       </j-write>
      </j-try>
      <j-catch param="err" data-catch>
        <j-write element="'#status'" method="write"
          output="err" data-write></j-write>
      </j-catch>

  </j-afx>
  <j-fx-call name="returnInstrument" param data-fx-call></j-fx-call>
</jh-script>
```

- **Fetch**
  - **Foundation**
    - Gets resource from a server whether it's a website containing a raw text data or a collection of text data.

```
<j-fetch data="filePath" data-fetch />
  <j-then data data-then><j-fx param="rawData" data-fx>
    <!--INSERT CODE HERE-->
  </j-fx></j-then>
  <j-then-close data data-then><j-fx param="realData" data-fx>
    <!--INSERT CODE HERE-->
  </j-fx></j-then-close>
```

- **Fetch**
  - **Working Example (Default)**
    - Here is a working code of the preview: (default implementation)

```
<p style="margin:0 0 8px 0;">Get a girl like a:</p>
  <img id="girlImage" style="max-height:300px;" alt="girl" hidden>
  <h2 id="girlDesc" style="margin:0;"></h2>

<jh-script>

  <j-fetch data="'https://api.npoint.io/296700a54ad7eda91d29'"
    data-fetch>
     <j-then data data-then><j-fx param="rawData" data-fx>
       <j-return value="rawData" data-json data-return></j-return>
     </j-fx></j-then>
     <j-then-close data data-then><j-fx param="realData" data-fx>
       <j-let data="imgPath" data-eq data-let>
         <j-value data="realData.girls[0].imagepath" data-value>
           </j-value>
       </j-let>
       <j-let data="imgDesc" data-eq data-let>
         <j-value data="realData.girls[0].description" data-value>
           </j-value>
       </j-let>

       <j-docget data-id="'girlImage'" data-prop="style.display"
         data-eq="'block'" data-docget-apply></j-docget>
       <j-docget data-id="'girlImage'" data-set-attr="'src'"
         data-attr-val="imgPath" data-docget-apply="attr"></j-docget>
       <j-write element="'#girlDesc'" method="write"
        output="imgDesc" data-write></j-write>
     </j-fx></j-then-close>
  </j-fetch>
</jh-script>
```

- **Fetch**
  - **Working Example (Async and Await)**
    - Here is a working code of the preview: (asynchronous implementation)

```
<p style="margin:0 0 8px 0;">Get a girl like a:</p>
  <img id="girlImage" style="max-height:300px;" alt="girl" hidden>
  <h2 id="girlDesc" style="margin:0;"></h2>

<jh-script>
 <j-afx name="getAGirl" param data-afx>
  <j-let data="rawData" data-eq data-let>
    <j-fetch data-await="'https://api.npoint.io/296700a54ad7eda91d29'"
      data-fetch />
  </j-let>
    <j-let data="realData" data-eq data-let>
      <j-value data-await="rawData" data-json data-value />
    </j-let>

    <j-let data="imgPath" data-eq data-let>
      <j-value data="realData.girls[0].imagepath" data-value></j-value>
    </j-let>
    <j-let data="imgDesc" data-eq data-let>
      <j-value data="realData.girls[0].description" data-value>
        </j-value>
     </j-let>

      <j-docget data-id="'girlImage'" data-prop="style.display"
        data-eq="'block'" data-docget-apply></j-docget>
      <j-docget data-id="'girlImage'" data-set-attr="'src'"
        data-attr-val="imgPath" data-docget-apply="attr"></j-docget>
      <j-write element="'#girlDesc'" method="write"
        output="imgDesc" data-write></j-write>
 </j-afx>
  <j-fx-call name="getAGirl" param data-fx-call></j-fx-call>

</jh-script>
```

- **Navigator**
  - **Methods**
    - **Cookies**
      - Checks if cookies enabled or not in the browser

```
<j-navigator type="cookieEnabled" data-navigator></j-navigator>
```

    - **Browser**
      - Returns the name of the browser

```
<j-navigator type="browser" data-navigator></j-navigator>
```

    - **Engine**
      - Returns the browser engine name

```
<j-navigator type="engine" data-navigator></j-navigator>
```

    - **Chromium**
      - Returns true if the browser is chromium based, else false

```
<j-navigator type="isChromium" data-navigator></j-navigator>
```

    - **Version**
      - Returns the browser version

```
<j-navigator type="version" data-navigator></j-navigator>
```

    - **Device Type**
      - Returns the device type

```
<j-navigator type="devType" data-navigator></j-navigator>
```

    - **Platform**
      - Returns the device platform

```
<j-navigator type="platform" data-navigator></j-navigator>
```

    - **Language**
      - Returns the language of the browser

```
<j-navigator type="language" data-navigator></j-navigator>
```

    - **Connectivity**
      - Returns true if the browser connectivity is online, else false

```
<j-navigator type="isConnected" data-navigator></j-navigator>
```

- **Altogether (Navigator)**
  - Here is the **preview** of all of the navigator methods:

  - Here is the **working code** of the preview:

```
<p style="margin:0;" id="navTest">
 <span style="color:red;"><b>Navigator</b>:</span>
</p>
<jh-script>
  <j-append element="'#navTest'" data-append-block>
    <j-value-rbr/>
    <j-value data="'<b>Cookies Enabled:</b> '" data-add data-value></j-value>
    <j-navigator type="cookieEnabled" data-navigator></j-navigator><j-value-br/>
    <j-value data="'<b>Browser:</b> '" data-add data-value></j-value>
    <j-navigator type="browser" data-navigator></j-navigator><j-value-br/>
    <j-value data="'<b>Engine:</b> '" data-add data-value></j-value>
    <j-navigator type="engine" data-navigator></j-navigator><j-value-br/>
    <j-value data="'<b>Chromium?:</b> '" data-add data-value></j-value>
    <j-navigator type="isChromium" data-navigator></j-navigator><j-value-br/>
    <j-value data="'<b>Version:</b> '" data-add data-value></j-value>
    <j-navigator type="version" data-navigator></j-navigator><j-value-br/>
    <j-value data="'<b>Device Type:</b> '" data-add data-value></j-value>
    <j-navigator type="devType" data-navigator></j-navigator><j-value-br/>
    <j-value data="'<b>Platform:</b> '" data-add data-value></j-value>
    <j-navigator type="platform" data-navigator></j-navigator><j-value-br/>
    <j-value data="'<b>Language:</b> '" data-add data-value></j-value>
    <j-navigator type="language" data-navigator></j-navigator><j-value-br/>
    <j-value data="'<b>Connected:</b> '" data-add data-value></j-value>
    <j-navigator type="isConnected" data-navigator></j-navigator>
  </j-append>
</jh-script>
```

- **Location**
  - **Methods**
    - **Href**
      - Returns the URL or HREF of the page

```
<j-loc method="location" type="href" data-loc></j-loc>
```

```
<p style="margin:0;" id="locTest"><b>LOCATION HREF: </b></p>
  <jh-script>
    <j-append element="'#locTest'" data-append-block>
     <j-loc method="location" type="href" data-loc></j-loc>
    </j-append>
</jh-script>
```

    - **Hostname**
      - Returns the domain name of the page

```
<j-loc method="location" type="hostname" data-loc></j-loc>
```

```
<p style="margin:0;" id="locTest"><b>LOCATION HOSTNAME: </b></p>
<jh-script>
   <j-append element="'#locTest'" data-append-block>
     <j-loc method="location" type="hostname" data-loc></j-loc>
   </j-append>
</jh-script>
```

- **Pathname**
  - Returns the path with filename if it has of the page

```
<j-loc method="location" type="pathname" data-loc></j-loc>
```

```
<p style="margin:0;" id="locTest"><b>LOCATION PATHNAME: </b></p>
<jh-script>
   <j-append element="'#locTest'" data-append-block>
     <j-loc method="location" type="pathname" data-loc></j-loc>
   </j-append>
</jh-script>
```

- **Protocol**
  - Returns if the web protocol is secure or not

```
<j-loc method="location" type="protocol" data-loc></j-loc>
```

```
<p style="margin:0;" id="locTest"><b>LOCATION PROCOL: </b></p>
<jh-script>
   <j-append element="'#locTest'" data-append-block>
     <j-loc method="location" type="protocol" data-loc></j-loc>
   </j-append>
</jh-script>
```

- **Port**
  - Returns the port number of the web protocol

```
<j-loc method="location" type="port" data-loc></j-loc>
```

```
<p style="margin:0;" id="locTest"><b>LOCATION PORT: </b></p>
<jh-script>
   <j-append element="'#locTest'" data-append-block>
     <j-loc method="location" type="port" data-loc></j-loc>
   </j-append>
</jh-script>
```

- **Assign**
  - Loads a document or a page

```
<j-loc method="location" type="assign"
  param="'https://www.google.com'" data-loc-action></j-loc>
```

```
<p style="margin:0;" id="locTest">
 <b>Assigning Page to Google (in 3 seconds)</b>
</p>
<jh-script>
  <j-set-time>
    <j-fx name="assignFunctAx" param data-fx>
      <j-loc method="location" type="assign"
        param="'https://www.google.com'" data-loc-action></j-loc>
     </j-fx>
     <j-timeout time="3000"></j-timeout>
  </j-set-time>
</jh-script>
```

  - This script will go to the **Google** search engine site in 3 seconds

- **Property**
  - This is exclusive on Jaihoe and it **needs** an **internet connection**, you can only **use** these methods **inside** an **asynchronous** function
    - **IP Address**
      - Returns the detected IP address

```
<p style="margin:0;" id="locTest"><b>IP: </b></p>
<jh-script>
  <j-afx name="testLoc" param data-afx>
    <j-append element="'#locTest'" data-append-block>
      <j-loc method="property" type="ip" data-loc></j-loc>
    </j-append>
  </j-afx>
  <j-fx-call name="testLoc" param data-fx-call></j-fx-call>
</jh-script>
```

    - **Internet Service Provider**
      - Returns the detected internet service provider

```
<p style="margin:0;" id="locTest"><b>ISP: </b></p>
<jh-script>
  <j-afx name="testLoc" param data-afx>
    <j-append element="'#locTest'" data-append-block>
      <j-loc method="property" type="isp" data-loc></j-loc>
    </j-append>
  </j-afx>
  <j-fx-call name="testLoc" param data-fx-call></j-fx-call>
</jh-script>
```

- **Country**
  - Returns the detected country

```
<p style="margin:0;" id="locTest"><b>Country: </b></p>
<jh-script>
 <j-afx name="testLoc" param data-afx>
   <j-append element="'#locTest'" data-append-block>
     <j-loc method="property" type="country" data-loc></j-loc>
   </j-append>
 </j-afx>
  <j-fx-call name="testLoc" param data-fx-call></j-fx-call>
</jh-script>
```

- **Postal Code**
  - Returns the detected postal code

```
<p style="margin:0;" id="locTest"><b>Postal Code: </b></p>
<jh-script>
 <j-afx name="testLoc" param data-afx>
   <j-append element="'#locTest'" data-append-block>
     <j-loc method="property" type="postal" data-loc></j-loc>
   </j-append>
 </j-afx>
  <j-fx-call name="testLoc" param data-fx-call></j-fx-call>
</jh-script>
```

- **Time Zone**
  - Returns the detected time zone

```
<p style="margin:0;" id="locTest"><b>Timezone: </b></p>
<jh-script>
 <j-afx name="testLoc" param data-afx>
   <j-append element="'#locTest'" data-append-block>
    <j-loc method="property" type="timezone" data-loc></j-loc>
   </j-append>
 </j-afx>
  <j-fx-call name="testLoc" param data-fx-call></j-fx-call>
</jh-script>
```

- **Latitude**
  - Returns the detected latitude

```
<p style="margin:0;" id="locTest"><b>Latitude: </b></p>
<jh-script>
 <j-afx name="testLoc" param data-afx>
   <j-append element="'#locTest'" data-append-block>
    <j-loc method="property" type="latitude" data-loc></j-loc>
   </j-append>
 </j-afx>
  <j-fx-call name="testLoc" param data-fx-call></j-fx-call>
</jh-script>
```

- **Longitude**
  - Returns the detected longitude

```
<p style="margin:0;" id="locTest"><b>Longitude: </b></p>
<jh-script>
 <j-afx name="testLoc" param data-afx>
  <j-append element="'#locTest'" data-append-block>
   <j-loc method="property" type="longitude" data-loc></j-loc>
  </j-append>
 </j-afx>
  <j-fx-call name="testLoc" param data-fx-call></j-fx-call>
</jh-script>
```

- **Update**
  - Updates the values of the properties, since **properties** are meant to be **static**, however **this function** sends a **request** to **fetch** the new data location (needs to be inside an asynchronous function)

```
<j-loc method="property" type="update" param="'locProperty'"
  data-loc-action></j-loc>
```

  - This is **initiated** by:

```
<j-afx name="testLoc" param data-afx>
 <j-write element="'#long'" data-write-block>
   <j-loc method="property" type="longitude" data-loc></j-loc>
 </j-write>
 <j-write element="'#lat'" data-write-block>
   <j-loc method="property" type="latitude" data-loc></j-loc>
 </j-write>

</j-afx>
 <j-fx-call name="testLoc" param data-fx-call></j-fx-call>
```

- **Update**
  - Here is the working code of the preview:

```
<p>POS: (<span id="long">0</span>,<span id="lat">0</span>)</p>
<button id="updateLocation">Update Location</button>
<jh-script>
 <j-afx name="testLoc" param data-afx>
  <j-write element="'#long'" data-write-block>
     <j-loc method="property" type="longitude" data-loc></j-loc>
   </j-write>
   <j-write element="'#lat'" data-write-block>
     <j-loc method="property" type="latitude" data-loc></j-loc>
   </j-write>
 </j-afx>
 <j-fx-call name="testLoc" param data-fx-call></j-fx-call>

 <j-add-evt element="#updateLocation" data-elem-evtype="click"
   data-add-evt>
   <j-afx name="reloadLocFx" param data-fx>

    <j-write element="'#long'" data-write-output="'0'" data-write>
   </j-write>
    <j-write element="'#lat'" data-write-output="'0'" data-write>
   </j-write>

   <j-write element="'#long'" data-write-block>
     <j-loc method="property" type="update" param="'longitude'"
       data-loc-action></j-loc>
   </j-write>
  <j-write element="'#lat'" data-write-block>
     <j-loc method="property" type="update" param="'latitude'"
       data-loc-action></j-loc>
   </j-write>

    </j-afx>
 </j-add-evt>
</jh-script>
```

- **Error**
  - **Try and Catch**
    - **Try**
      - A block that examines all code inside, if an error occurs it fall to the catch block

```
<j-try>
</j-try>
```

    - **Catch**
      - A block that catches the error within the try block which is written above it, and this block has a error parameter on it

```
<j-catch param="err" data-catch>
</j-catch>
```

❖ Try and Catch

➢ Here is the preview and working code of try and catch

```
<p>(Error) <span style="color:red;" id="errCode"></span></p>
<!-- JaihoeScript Code -->
<jh-script>
  <j-try>
    <j-fx-call name="thisFx" param="" data-fx-call></j-fx-call>
  </j-try>
  <j-catch param="err" data-catch>
    <j-write element="'#errCode'" method="write"
       output="err" data-write></j-write>
  </j-catch>
</jh-script>
```

❖ Throw Statement

➢ Defines an error inside a try block and catches to the catch block

```
<j-throw data="'empty'" data-throw></j-throw>
```

➢ See the code on the next page, and try it.

- **Throw Statement**
  - Defines an error inside a try block and catches to the catch block

```
<p>Please input a number from 5 to 10:</p>
  <input id="numHold" type="text">
  <button id="testNum" type="button">Test Input</button>
  <p style="color:red;" id="errCode"></p>

<jh-script>
  <j-add-evt element="#testNum" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
       <j-write element="'#errCode'" method="write"
         output="''" data-write></j-write>
       <j-let data="x" data-eq data-let>
         <j-docget data-id="'numHold'" data-prop="value"
           data-docget-extend></j-docget>
       </j-let>
       <j-try>
          <j-if data="x.trim() == ''" data-if>
            <j-throw data="'empty'" data-throw></j-throw>
          </j-if>
          <j-if data="isNaN(x)" data-if>
            <j-throw data="'not a number'" data-throw></j-throw>
          </j-if>
          <j-if data="Number(x) < 5" data-if>
            <j-throw data="'too low'" data-throw></j-throw>
          </j-if>
          <j-if data="Number(x) > 10" data-if>
            <j-throw data="'too high'" data-throw></j-throw>
          </j-if>

       </j-try>
       <j-catch param="err" data-catch>
         <j-write element="'#errCode'" method="write"
            output="'Input is '+err" data-write></j-write>
       </j-catch>
    </j-fx>
  </j-add-evt>
</jh-script>
```

- **Finally Statement**
  - A block that executes a code after try and catch execution

```
<j-finally>
</j-finally>
```

  - See the code on the next page and try it.

- **Finally Statement**
  - Here is the preview and working code of finally statement included in the code

```
<p>Please input a number from 5 to 10:</p>
  <input id="numHold" type="text">
  <button id="testNum" type="button">Test Input</button>
  <p style="color:red;" id="errCode"></p>

<jh-script>
  <j-add-evt element="#testNum" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
       <j-write element="'#errCode'" method="write"
         output="''" data-write></j-write>
       <j-let data="x" data-eq data-let>
         <j-docget data-id="'numHold'" data-prop="value"
           data-docget-extend></j-docget>
       </j-let>
       <j-try>
          <j-if data="x.trim() == ''" data-if>
            <j-throw data="'empty'" data-throw></j-throw>
          </j-if>
          <j-if data="isNaN(x)" data-if>
            <j-throw data="'not a number'" data-throw></j-throw>
          </j-if>
          <j-if data="Number(x) < 5" data-if>
            <j-throw data="'too low'" data-throw></j-throw>
          </j-if>
          <j-if data="Number(x) > 10" data-if>
            <j-throw data="'too high'" data-throw></j-throw>
          </j-if>

       </j-try>
       <j-catch param="err" data-catch>
         <j-write element="'#errCode'" method="write"
           output="'Input is '+err" data-write></j-write>
       </j-catch>
       <j-finally>
          <j-docget data-id="'numHold'" data-prop="value"
            data-eq="''" data-docget-apply></j-docget>
       </j-finally>
    </j-fx>
  </j-add-evt>
</jh-script>
```

  - After **validation**, the input will be cleared.

- **Alert**
  - **Legacy**
    - Displays the old alert box with a message and a confirmation button

```
<div style="margin-top:110px;">
   <p style="margin:0;display:inline;">Legacy Alert:</p>
   <button id="showAlert" type="button">Show Alert</button>
 </div>

<jh-script>
  <j-add-evt element="#showAlert" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-alert type="legacy" param="'Hello World'"
         data-alert></j-alert>
     </j-fx>
  </j-add-evt>
</jh-script>
```

  - **Alert Notification**
    - Displays an alert box like a notification below the bottom left side with the default properties

- **Default**
  - Here is the working code of the preview

```
<p style="margin:0;display:inline;">New Alert:</p>
    <button id="showAlert" type="button">Show Alert</button>

<jh-script>
   <j-add-evt element="#showAlert" data-elem-evtype="click"
     data-add-evt>
      <j-fx param data-fx>
        <j-alert param="'Hello World'" data-alert></j-alert>
      </j-fx>
   </j-add-evt>
</jh-script>
```

  - For string input: `param="'Text'"` will show **Text**
  - For number input: `param="555"` will show **555**
  - For array input: `param="['Hello ','World']"` will show **Hello World**

- **Alert Notification Block**
  - **Default Example**
    - Displays the new alert box displayed below the bottom left side with custom properties

```
<p style="margin:0;display:inline;">New Alert Block:</p>
<button id="showAlert" type="button">Show Alert</button>

<jh-script>
  <j-add-evt element="#showAlert" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
      <j-alert-block>
        <j-attr name="value" data-value-sq="Hello guys!"
          data-attr-sq-close></j-attr>
      </j-alert-block>
    </j-fx>
  </j-add-evt>
</jh-script>
```

- **Other Alert Block Properties**
  - You can add more properties to the alert box
    - **Type**
      - This the theme of the alert box
        - **Default**

```
<p style="margin:0;display:inline;">New Alert Block:</p>
<button id="showAlert" type="button">Show Alert</button>

<jh-script>
  <j-add-evt element="#showAlert" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
      <j-alert-block>
        <j-attr name="type" data-value-sq="success"
          data-attr-sq></j-attr>
        <j-attr name="value" data-value-sq="Downloaded!"
          data-attr-sq-close></j-attr>
      </j-alert-block>
    </j-fx>
  </j-add-evt>
</jh-script>
```

        - You can replace the type **(success)** with **info, warning, danger**; **Example:** `name="type" data-value-sq="danger"`

        - **Custom**
          - You can customize the alert box with 3 more properties

- **Custom**
  - Here is the working code of the preview

```
<p style="margin:0;display:inline;">New Alert Block:</p>
<button id="showAlert" type="button">Show Alert</button>

<jh-script>
  <j-add-evt element="#showAlert" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
      <j-alert-block>
        <j-attr name="type" data-value-sq="custom"
          data-attr-sq></j-attr>
        <j-attr name="value" data-value-sq="Merry Christmas!"
          data-attr-sq></j-attr>
        <j-attr name="textColor" data-value-sq="red"
          data-attr-sq></j-attr>
        <j-attr name="bgColor" data-value-sq="yellow"
          data-attr-sq></j-attr>
        <j-attr name="borderColor" data-value-sq="green"
          data-attr-sq-close></j-attr>
      </j-alert-block>

    </j-fx>
  </j-add-evt>
</jh-script>
```

  - You just have to **additionally define** the following: textColor (text color), bgColor (background color), borderColor (border color)
  - Don't worry, if the values are incorrect, it has a default value at least

- **Delay**
  - The opening time of the alert box

- **Delay**
  - Here is the working code of the preview:

```
<p style="margin:0;display:inline;">
 New Alert Block show in <b>(3 seconds)</b> below:
</p>
<jh-script>
  <j-alert-block>
    <j-attr name="value" data-value-sq="Suprise!"
      data-attr-sq></j-attr>
    <j-attr name="delay" data-value-sq="3000"
      data-attr-sq-close></j-attr>
  </j-alert-block>
</jh-script>
```

  - The alert box will be shown in 3 seconds.
  - If you want to show the alert box **immediately**, just **remove** the **delay attribute** or set the **delay value** to **false**
    `name="delay" data-value-sq="false"`

- **Timeout**
  - The closing time of the alert box

```
<p style="margin:0;display:inline;">
 New Alert Block be gone in <b>(5 seconds)</b> below:
</p>
<jh-script>
 <j-alert-block>
   <j-attr name="value" data-value-sq="Please wait!"
     data-attr-sq></j-attr>
   <j-attr name="timeout" data-value-sq="5000" data-attr-sq-close>
     </j-attr>
 </j-alert-block>
</jh-script>
```

  - The alert box will be gone in **5 seconds**
  - To make the alert box **stay forever**, just set the **timeout value** to **false**:
    `name="timeout" data-value-sq="false"`

- **Dismissible**
  - Adds a closing feature for the alert box

```
<p style="margin:0;display:inline;">Please click the alert box!</p>
<jh-script>
  <j-alert-block>
    <j-attr name="value" data-value-sq="Click to dismiss!"
      data-attr-sq></j-attr>
    <j-attr name="timeout" data-value-sq="false" data-attr-sq>
      </j-attr>
    <j-attr name="dismiss" data-value-sq="true" data-attr-sq-close>
      </j-attr>
  </j-alert-block>
</jh-script>
```

  - Just add the **dismiss attribute** and set the value to **true**
    `name="dismiss" data-value-sq="true"`
  - By clicking the alert box, the alert box will be gone

- **Dismissible Button**
  - Adds a closing feature for the alert box by clicking the close button

- **Dismissible Button**
  - Here is the working code of the preview

```
<p style="margin:0;display:inline;">Please click the close
button!</p>
<jh-script>
  <j-alert-block>
    <j-attr name="value" data-value-sq="Click to dismiss!"
      data-attr-sq></j-attr>
    <j-attr name="timeout" data-value-sq="false" data-attr-sq>
      </j-attr>
    <j-attr name="dismiss" data-value-sq="true" data-attr-sq>
      </j-attr>
    <j-attr name="dismissBTN" data-value-sq="true"
     data-attr-sq-close></j-attr>
  </j-alert-block>
</jh-script>
```

- **Alert Dialog**
  - Displays a new dialog alert box with a message and a confirmation button, it does not return any value and it has only one liner message

```
<j-alert type="dialog" param="'Enjoy your stay here!'"
  data-alert></j-alert>
```

  - For string input: `param="'Text'"` will show **Text**
  - For number input: `param="555"` will show **555**
  - For array input: `param="['Hello ','World']"` will show **Hello World**

- **Alert Dialog**
  - Here is the working code of the preview:

```
<p style="margin:0;display:inline;">Jake's Home: </p>
<button id="showAlertD" type="button">Get Started</button>

<jh-script>
  <j-add-evt element="#showAlertD" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-alert type="dialog" param="'Enjoy your stay here!'"
         data-alert></j-alert>
     </j-fx>
  </j-add-evt>
</jh-script>
```

- **Alert Dialog Block**
  - Displays a new dialog alert box with a message (title and description) and a confirmation button, it does not return any value but you can customize it

```
<j-alert-dialog>
  <!--INSERT CODE HERE-->
</j-alert-dialog>
```

- **Other Alert Dialog Block Properties**
  - **Default**

```
<j-attr name="display" data-value-sq="Title!"
  data-attr-sq></j-attr>
<j-attr name="subdisplay" data-value-sq="Subtitle."
  data-attr-sq></j-attr>
```

  - **Change confirm button value**

```
<j-attr name="confirmText"  data-value-sq="Noted"
  data-attr-sq></j-attr>
```

  - **Change confirm button color**

```
<j-attr name="colorTheme" data-value-sq="#FF4081"
  data-attr-sq></j-attr>
```

- **Alert Dialog Block**
  - Here is the working code of the preview

```
<p style="margin:0;display:inline;">John's Resto: </p>
<button id="showAlertD" type="button">Get Started</button>

<jh-script>
 <j-add-evt element="#showAlertD" data-elem-evtype="click"
  data-add-evt>
  <j-fx param data-fx>
   <j-alert-dialog>
    <j-attr name="display" data-value-sq="Welcome to our Restaurant!"
      data-attr-sq></j-attr>
    <j-attr name="subdisplay"
      data-value-sq="Try to select a food that you want to order."
      data-attr-sq></j-attr>
   </j-alert-dialog>
  </j-fx>
 </j-add-evt>
</jh-script>
```

- **Prompt**
  - **Legacy**
    - Display an old dialog box that gets user input, for a function

```
<j-prompt type="legacy"
  param="'Please enter your name','Jerwin'" data-prompt></j-prompt>
```

```
<!-- JaihoeScript Code -->
<p style="margin:0;"><b>Legacy Prompt Test:</b></p>
<p id="personGreet" style="margin:0;"></p>
<jh-script>
   <j-let data="person" data-eq data-let>
     <j-prompt type="legacy"
       param="'Please enter your name','Jerwin'" data-prompt>
        </j-prompt>
   </j-let>

   <j-if data="person != null" data-if>
     <j-write element="'#personGreet'" method="write"
       output="'Hello '+person+'! how are you today?'"
       data-write></j-write>
   </j-if>
</jh-script>
```

- **Prompt**
  - **Prompt Block**
    - This is exclusive only on Jaihoe, displays a new customizable dialog box that gets user input, you can only **use** this method **inside** an **asynchronous** function

```
<j-prompt-block>
  <!--INSERT ATTRIBUTES HERE-->
</j-prompt-block>
```

```
<p style="margin:0;display:inline;"><b>NEW Prompt Test:</b></p>
<button id="showPrompt" type="button">Show Prompt!</button>
<p id="personGreet" style="margin:5px 0 0 0;"></p>

<jh-script>
  <j-add-evt element="#showPrompt" data-elem-evtype="click"
    data-add-evt>
     <j-afx param data-afx>

       <j-let data="person" data-eq data-let>
         <j-prompt-block>
          <j-attr name="display"
            data-value-sq="What is your name?"
            data-attr-sq></j-attr>
          <j-attr name="input" data-value-sq="Jerwin"
            data-attr-sq></j-attr>
         </j-prompt-block>
       </j-let>
       <j-if data="person != null" data-if>
       <j-write element="'#personGreet'" method="write"
         output="'Hello '+person+'! how are you today?'"
         data-write></j-write>
       </j-if>
     </j-afx>
  </j-add-evt>
</jh-script>
```

    - You could also use **prompt** in **one line**, but only the **display** attribute will be changed

```
<j-prompt param="'Whats your name?'" data-prompt></j-prompt>
```

      - Its not that **customizable** so prompt-block is better to use

- **Prompt Block Attributes**
  - **Default Values**

```
<j-attr name="display" data-value-sq="What is your name?"
  data-attr-sq></j-attr>
<j-attr name="input" data-value-sq="Jerwin"
  data-attr-sq></j-attr>
```

  - **To add a placeholder**

```
<j-attr name="placeholder" data-value-sq="Jerwin"
  data-attr-sq></j-attr>
```

  - **To change button values**

```
<j-attr name="cancelText" data-value-sq="exit"
  data-attr-sq></j-attr>
<j-attr name="confirmText" data-value-sq="go"
  data-attr-sq></j-attr>
```

  - **To remove the cancel button**

```
<j-attr name="cancelBTN" data-value-sq="false"
  data-attr-sq></j-attr>
```

  - **To change color theme**

```
<j-attr name="colorTheme" data-value-sq="#FF4081"
  data-attr-sq></j-attr>
```

- **Confirm**
  - **Legacy**
    - Display an old dialog box with a condition statement and condition options, for a function

```
<p style="margin:0;display:inline;"><b>Legacy Confirm Test:</b></p>
<p id="status" style="margin:5px 0 0 0;"></p>
<jh-script>
   <j-let data="action" data-eq data-let>
    <j-confirm type="legacy"
      param="'Do you want to receive your order?'" data-confirm>
    </j-confirm>
   </j-let>
   <j-if data="action == true" data-if>
     <j-write element="'#status'" method="write"
       output="'Ok, serving your order!'" data-write></j-write>
   </j-if>
   <j-else>
     <j-write element="'#status'" method="write"
       output="'Ok, canceling your order!'" data-write></j-write>
   </j-else>
</jh-script>
```

- **Confirm**
  - **Confirm Block**
    - This is exclusive only on **Jaihoe**, displays a new customizable dialog box with a condition statement (consists of title and description), additionally the condition options, you can only **use** this method **inside** an **asynchronous** function

```
<p style="margin:0;display:inline;"><b>NEW Confirm Test:</b></p>
<button id="showConfirm" type="button">Start to order!</button>
<p id="status" style="margin:5px 0 0 0;"></p>

<jh-script>
  <j-add-evt element="#showConfirm" data-elem-evtype="click"
    data-add-evt>
    <j-afx param data-afx>
      <j-let data="action" data-eq data-let>
        <j-confirm-block>
          <j-attr name="display" data-value-sq="Receive Food Order?"
            data-attr-sq></j-attr>
          <j-attr name="subdisplay"
            data-value-sq="Do you want to receive your order?"
            data-attr-sq></j-attr>
        </j-confirm-block>
       </j-let>
       <j-if data="action == true" data-if>
         <j-write element="'#status'" method="write"
           output="'Ok, serving your order!'" data-write></j-write>
       </j-if>
       <j-else>
         <j-write element="'#status'" method="write"
           output="'Ok, canceling your order!'" data-write></j-write>
       </j-else>
     </j-afx>
  </j-add-evt>
</jh-script>
```

    - You could also use **confirm** in **one line**, but only the **display** attribute will be changed

```
<j-confirm param="'Receive order?'" data-confirm></j-confirm>
```

    - Its not that **customizable** so confirm-block is better to use

- **Confirm Block Attributes**
  - **Default Values**

```
<j-attr name="display" data-value-sq="Receive Food Order?"
  data-attr-sq></j-attr>
<j-attr name="subdisplay"
  data-value-sq="Do you want to receive your order?"
  data-attr-sq></j-attr>
```

  - **To change button values**

```
<j-attr name="cancelText" data-value-sq="n"
  data-attr-sq></j-attr>
<j-attr name="confirmText" data-value-sq="yes"
  data-attr-sq></j-attr>
```

  - **To change the color of the confirm button**

```
<j-attr name="colorTheme" data-value-sq="#FF4081"
  data-attr-sq></j-attr>
```

- **Clipboard**
  - Its all about automatic copying, cutting and pasting text instead of manually do it

- **Copy**
  - **Internal (Async/Await Return Value)**
    - If you want to copy to clipboard with return value
    - Please enter only string or number for the data attribute

```
<j-await/><j-copy data="'Copy text from internal script'"
  data-copy></j-copy>
```

```
<p style="display:inline;border:1px solid blue;">
 Copy text from internal script
</p>
<button id="copyInt" type="button">Copy text</button><br/>
<input type="text" placeholder="paste text here">

<jh-script>

 <j-add-evt element="#copyInt" data-elem-evtype="click"
   data-add-evt>
   <j-afx param data-afx>
     <j-let data="hasCopy" data-eq data-let>
       <j-await/><j-copy data="'Copy text from internal script'"
         data-copy></j-copy>
     </j-let>
     <j-if data="hasCopy == true" data-if>
       <j-alert param="'Copied to clipboard!'"
         data-alert></j-alert>
       </j-if>
    </j-afx>
 </j-add-evt>
</jh-script>
```

- **Copy**
  - **Internal (No Return Value)**
    - If you just want to copy a text without a return value
    - Please enter only string or number for the data attribute

```
<j-copy data="'Copy text from internal script'"
  data-copy="block"></j-copy>
```

```
<p style="display:inline;border:1px solid blue;">
  Copy text from internal script
</p>
<button id="copyInt" type="button">Copy text</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
   <j-add-evt element="#copyInt" data-elem-evtype="click"
     data-add-evt>
      <j-fx param data-fx>
        <j-copy data="'Copy text from internal script'"
          data-copy="block"></j-copy>
        <j-alert param="'Trying to copy to clipboard!'"
          data-alert></j-alert>
      </j-fx>
   </j-add-evt>
</jh-script>
```

- **Copy**
  - **TextArea Method (Async/Await Return Value)**
    - If you want to copy a text from the textarea with return value

```
<j-await/><j-copy element="#contentTextArea"
  data-copy="element"></j-copy>
```

- **Copy**
  - **TextArea Method (Async/Await Return Value)**
    - Here is the working code of the preview

```
<textarea id="contentTextArea">Copy text from textarea</textarea>
<button id="copyInput" type="button">Copy textarea</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
  <j-add-evt element="#copyInput" data-elem-evtype="click"
    data-add-evt>
     <j-afx param data-afx>
       <j-let data="hasCopy" data-eq data-let>
         <j-await/><j-copy element="#contentTextArea"
           data-copy="element"></j-copy>
       </j-let>
       <j-if data="hasCopy == true" data-if>
         <j-alert param="'Copied to clipboard!'" data-alert>
           </j-alert>
       </j-if>
     </j-afx>
  </j-add-evt>
</jh-script>
```

- **Copy**
  - **TextArea Method (No Return Value)**
    - If you just want to copy a text from the text area without return value

```
<j-copy element="#contentTextArea" data-copy="element-block">
  </j-copy>
```

```
<textarea id="contentTextArea">Copy text from textarea</textarea>
<button id="copyInput" type="button">Copy textarea</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
 <j-add-evt element="#copyInput" data-elem-evtype="click"
   data-add-evt>
  <j-fx param data-fx>
   <j-copy element="#contentTextArea" data-copy="element-block">
     </j-copy>
   <j-alert param="'Trying to copy to clipboard!'" data-alert>
     </j-alert>
  </j-fx>
 </j-add-evt>
</jh-script>
```

- **Copy**
  - **InputBox Method (Async/Await Return Value)**
    - If you want to copy a text from the input box with return value

```
<j-await/><j-copy element="#contentInput" data-copy="element">
  </j-copy>
```

```
<input id="contentInput" type="text" value="Copy text from input">
<button id="copyInput" type="button">Copy input box</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
  <j-add-evt element="#copyInput" data-elem-evtype="click"
    data-add-evt>
     <j-afx param data-afx>
       <j-let data="hasCopy" data-eq data-let>
         <j-await/><j-copy element="#contentInput"
           data-copy="element"></j-copy>
       </j-let>
       <j-if data="hasCopy == true" data-if>
         <j-alert param="'Copied to clipboard!'" data-alert>
           </j-alert>
       </j-if>
     </j-afx>
  </j-add-evt>
</jh-script>
```

- **Copy**
  - **InputBox Method (No Return Value)**
    - If you just want to copy a text from the input box without return value

```
<j-copy element="#contentInput" data-copy="element-block"></j-copy>
```

- **Copy**
  - **InputBox Method (No Return Value)**
    - Here is the working code of the preview

```
<input id="contentInput" type="text" value="Copy text from input">
<button id="copyInput" type="button">Copy input box</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
  <j-add-evt element="#copyInput" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
        <j-copy element="#contentInput"
          data-copy="element-block"></j-copy>
       <j-alert param="'Trying to copy to clipboard!'" data-alert>
         </j-alert>
     </j-fx>
  </j-add-evt>
</jh-script>
```

- **Copy**
  - **Other Elements (Async/Await Return Value)**
    - If you want to copy a text from other elements with return value

```
<j-await/><j-copy element="#thisHtmlContent"
  data-copy="element"></j-copy>
```

```
<p style="display:inline;border:1px solid red;"
 id="thisHtmlContent">Copy text from html content</p>
<button id="copyHTMLContent" type="button">Copy text</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
  <j-add-evt element="#copyHTMLContent" data-elem-evtype="click"
    data-add-evt>
     <j-afx param data-afx>
       <j-let data="hasCopy" data-eq data-let>
         <j-await/><j-copy element="#thisHtmlContent"
           data-copy="element"></j-copy>
       </j-let>
       <j-if data="hasCopy == true" data-if>
         <j-alert param="'Copied to clipboard!'" data-alert>
           </j-alert>
       </j-if>
     </j-afx>
  </j-add-evt>
</jh-script>
```

- **Copy**
  - **Other Elements (No Return Value)**
    - If you just want to copy a text from other elements without return value

```
<j-copy element="#thisHtmlContent" data-copy="element-block">
  </j-copy>
```

```
<p style="display:inline;border:1px solid red;"
 id="thisHtmlContent">Copy text from html content</p>
<button id="copyHTMLContent" type="button">Copy text</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
 <j-add-evt element="#copyHTMLContent" data-elem-evtype="click"
  data-add-evt>
  <j-fx param data-fx>
   <j-copy element="#thisHtmlContent" data-copy="element-block">
    </j-copy>
   <j-alert param="'Trying to copy to clipboard!'" data-alert>
    </j-alert>
  </j-fx>
 </j-add-evt>
</jh-script>
```

- **Cut**
  - **Text Area Method (Async/Await Return Value)**
    - If you want to cut a text from the text area with return value

```
<j-await/><j-cut element="#contentTextArea" data-cut="element">
  </j-cut>
```

- **Cut**
  - **Text Area Method (Async/Await Return Value)**
    - Here is the working code of the preview

```
<textarea id="contentTextArea">Cut text from textarea</textarea>
<button id="cutInput" type="button">cut textarea</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
  <j-add-evt element="#cutInput" data-elem-evtype="click"
    data-add-evt>
     <j-afx param data-afx>
       <j-let data="hasCut" data-eq data-let>
         <j-await/><j-cut element="#contentTextArea"
           data-cut="element"></j-cut>
       </j-let>
       <j-if data="hasCut == true" data-if>
         <j-alert param="'Copied to clipboard!'" data-alert>
           </j-alert>
       </j-if>
     </j-afx>
  </j-add-evt>
</jh-script>
```

- **Cut**
  - **Text Area Method (No Return Value)**
    - If you just want to cut a text from the text area without return value

```
<j-cut element="#contentTextArea" data-cut="element-block"></j-cut>
```

```
<textarea id="contentTextArea">Cut text from textarea</textarea>
<button id="cutInput" type="button">Cut textarea</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
  <j-add-evt element="#cutInput" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
      <j-cut element="#contentTextArea" data-cut="element-block">
        </j-cut>
      <j-alert param="'Trying to copy to clipboard!'" data-alert>
        </j-alert>
    </j-fx>
  </j-add-evt>
</jh-script>
```

- **Cut**
  - **Text Box Method (Async/Await Return Value)**
    - If you want to cut a text from the text box with return value

```
<j-await/><j-cut element="#contentInput" data-cut="element"></j-cut>
```

```
<input id="contentInput" type="text" value="Cut text from input">
<button id="cutInput" type="button">Cut input box</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
  <j-add-evt element="#cutInput" data-elem-evtype="click"
     data-add-evt>
     <j-afx param data-afx>
        <j-let data="hasCut" data-eq data-let>
          <j-await/><j-cut element="#contentInput"
            data-cut="element"></j-cut>
        </j-let>
        <j-if data="hasCut == true" data-if>
          <j-alert param="'Copied to clipboard!'" data-alert>
            </j-alert>
        </j-if>
      </j-afx>
  </j-add-evt>
</jh-script>
```

- **Cut**
  - **Text Box Method (No Return Value)**
    - If you just want to cut a text from the text box without return value

- **Cut**
  - **Text Box Method (No Return Value)**
    - Here is the working code of the preview

```
<input id="contentInput" type="text" value="Cut text from input">
<button id="cutInput" type="button">Cut input box</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
  <j-add-evt element="#cutInput" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
        <j-cut element="#contentInput"
          data-cut="element-block"></j-cut>
       <j-alert param="'Trying to copy to clipboard!'" data-alert>
         </j-alert>
     </j-fx>
  </j-add-evt>
</jh-script>
```

- **Cut**
  - **Other Element (Async/Await Return Value)**
    - If you want to cut a text from other elements with return value

    - If you noticed, only the **text was gone** in the paragraph.
    - However there are **conditions**:
      - If you target it to a text element (like **paragraph** or **header** tags), all content will be gone
      - If you target it to other elements (like **div**, list), only the text will be gone, (Even Jaihoe tries to maintain the contents of the element)
        - If the text still **exist**, maybe there is a non-text formatting element inside the block, make sure it's a text formatting element like (bold, italic, span, etc.)

- **Cut**
  - **Other Element (Async/Await Return Value)**
    - Here is the working code of the preview

```
<div style="display: inline-block;border:1px solid black;">
 <div style="display: flex; align-items: center;">
   <div>
    <img src="https://jaihoejs.github.io/sample/img/verift.jpg" alt="sample">
   </div>
   <div><p id="thisHtmlContent" style="margin:0;display:inline;">Cut from <span
    style="color:red;">text</span> from HTML content</p></div>
  </div>
 </div>
 <div style="display: inline-block;">
  <button style="display:inline;" id="cutInput" type="button">Cut text</button>
 </div>
 <br/>
 <input type="text" placeholder="paste text here">

<jh-script>
  <j-add-evt element="#cutInput" data-elem-evtype="click" data-add-evt>
    <j-afx param data-afx>
      <j-let data="hasCut" data-eq data-let>
        <j-await/><j-cut element="#thisHtmlContent" data-cut="element"></j-cut>
      </j-let>
      <j-if data="hasCut == true" data-if>
        <j-alert param="'Copied to clipboard!'" data-alert></j-alert>
      </j-if>
    </j-afx>
  </j-add-evt>
</jh-script>
```

- **Cut**
  - **Other Element (No Return Value)**
    - If you just want to cut a text from other elements without return value

- **Cut**
  - **Other Element (No Return Value)**
    - Here is the working code of the preview

```
<div style="display: inline-block;border:1px solid black;">
  <div style="display: flex; align-items: center;">
   <div>
     <img src="https://jaihoejs.github.io/sample/img/verift.jpg" alt="sample">
   </div>
   <div><p id="thisHtmlContent" style="margin:0;display:inline;">Cut from <span
     style="color:red;">text</span> from HTML content</p></div>
  </div>
</div>
<div style="display: inline-block;">
 <button style="display:inline;" id="cutInput" type="button">Cut text</button>
</div>
<br/>
<input type="text" placeholder="paste text here">

<jh-script>
 <j-add-evt element="#cutInput" data-elem-evtype="click" data-add-evt>
   <j-fx param data-fx>
     <j-cut element="#thisHtmlContent" data-cut="element-block"></j-cut>
      <j-alert param="'Trying to copy to clipboard!'" data-alert></j-alert>
   </j-fx>
 </j-add-evt>
</jh-script>
```

- **Cut**
  - **Internal Method**
    - You can still copy the text but you cannot cut the text practically so it will show a warning in the console log, its not advisable to use it

```
<j-await/><j-cut data="'Cut text from internal script'"
  data-cut></j-cut>
```

- **Cut**
  - **Internal Method**
    - Here is the working code of the preview:

```
<p style="display:inline;border:1px solid blue;">Copy text from
internal script</p>
<button id="cutInt" type="button">Copy text</button><br/>
<input type="text" placeholder="paste text here">
<jh-script>
 <j-add-evt element="#cutInt" data-elem-evtype="click"
   data-add-evt>
   <j-afx param data-afx>
     <j-let data="hasCut" data-eq data-let>
       <j-await/><j-cut data="'Cut text from internal script'"
         data-cut></j-cut>
     </j-let>
     <j-if data="hasCut == true" data-if>
       <j-alert param="'Copied to clipboard!'"
         data-alert></j-alert>
       </j-if>
    </j-afx>
 </j-add-evt>
</jh-script>
```

- **Paste**
  - If you want to paste text to elements

```
<j-paste element="#element" data-paste="element-block"></j-paste>
```

  - **TextArea Element**
    - If you want to paste a text in a TextArea

```
<textarea id="pasteHolder" placeholder="Try to click paste
text"></textarea>
<button id="pasteTest" type="button">Paste text</button><br/>

<jh-script>
  <j-add-evt element="#pasteTest" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
        <j-copy data="'Pasted text to textarea!'"
          data-copy="block"></j-copy>
        <j-paste element="#pasteHolder" data-paste="element-block">
          </j-paste>
    </j-fx>
   </j-add-evt>
</jh-script>
```

- **Input Box Element**
  - If you want to paste a text in a Textbox

```
<input type="text" id="pasteHolder" placeholder="Try to click paste
text">
<button id="pasteTest" type="button">Paste text</button><br/>
<jh-script>
  <j-add-evt element="#pasteTest" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
        <j-copy data="'Pasted text to textbox!'"
          data-copy="block"></j-copy>
        <j-paste element="#pasteHolder"
          data-paste="element-block"></j-paste>
    </j-fx>
   </j-add-evt>
</jh-script>
```

- **Other Elements**
  - If you want to paste a text in in a certain HTML element

```
<p style="display:inline;margin:0;border:1px solid red;"
id="pasteHolder">Try to click paste text</p><br/>
<button style="margin-top:5px;" id="pasteTest" type="button">Paste
text</button><br/>
<jh-script>
  <j-add-evt element="#pasteTest" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
      <j-copy data="'Pasted text to HTML element!'"
        data-copy="block"></j-copy>
      <j-paste element="#pasteHolder"
        data-paste="element-block"></j-paste>
    </j-fx>
   </j-add-evt>
</jh-script>
```

- **Fullscreen**
  - **Request**
    - Literally makes a request to make a selected element full screen
    - **Example #1**: Entering Fullscreen

```
<video style="max-width: 100%;height: auto;" width="640" controls
id="nfsVideo">
  <source src="https://jaihoejs.github.io/sample/video/clouds.mp4"
type="video/mp4"></video><br/>
<button id="enterFS">Open Video in Fullscreen Mode</button>
<jh-script>
  <j-add-evt element="#enterFS" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
        <j-fullscreen method="request" element="#nfsVideo"
          data-fullscreen="element-block"></j-fullscreen>
     </j-fx>
   </j-add-evt>
</jh-script>
```

    - **Example #2**: Entering Document Fullscreen

```
<p style="margin:0;">Click the button to enter fullscreen!</p>
<button id="enterFS">Enter Fullscreen Mode</button>
<jh-script>
  <j-add-evt element="#enterFS" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
        <j-fullscreen method="request"></j-fullscreen>
    </j-fx>
   </j-add-evt>
</jh-script>
```

- **Fullscreen**
  - **Exit**
    - If you send a full screen request, you can also make an exit full screen request. Take note that it only works if you trigger a full screen request not by pressing keyboard

```
<p style="margin:0;">Click the enter or exit fullscreen!</p>
<button id="enterFS">Enter Fullscreen Mode</button>
<button id="exitFS">Exit Fullscreen Mode</button>

<jh-script>
  <j-add-evt element="#enterFS" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
        <j-fullscreen method="request"></j-fullscreen>
    </j-fx>
   </j-add-evt>

   <j-add-evt element="#exitFS" data-elem-evtype="click"
    data-add-evt>
    <j-fx param data-fx>
        <j-fullscreen method="exit"></j-fullscreen>
    </j-fx>
   </j-add-evt>
</jh-script>
```

- **Is Fullscreen**
  - Check if the page is in full screen depending on its case, there are two types of detecting full screen
    - **Detect**
      - It will just check whether if you enter in full screen or not regardless of the page being focused
    - **Active**
      - It will check if you trigger a full screen request that makes you in full screen, it does not detect keyboard shortcuts in entering full screen

- **Is Fullscreen (Working Code)**
  - **Detect**
    - Here is the working example if its fullscreen using detect method:

```
<p style="margin:0px;">
 Full Screen (Detect): <span id="fsOut"></span>
</p>
<p style="margin:0px;">
 To enter fullscreen, try to press <b>F11</b>
</p>
<p style="margin:0px;">
 To exit fullscreen, try to <b>swipe above the screen</b>
</p>

<jh-script>
    <j-write element="'#fsOut'" data-write-block>
        <j-isfullscreen method="detect"/>
    </j-write>

  <j-add-evt var="window" data-evtype="resize"
    data-add-evt>
    <j-fx param data-fx>
        <j-write element="'#fsOut'" data-write-block>
            <j-isfullscreen method="detect" />
        </j-write>

    </j-fx>
   </j-add-evt>
</jh-script>
```

- **Active**
  - Here is the working example if its fullscreen using active method:

```
<p style="margin:0;">
  Is FullScreen (Active):<span id="ifCFS">false</span>
</p>
<button id="enterFS">Enter Fullscreen Mode</button>
<button id="exitFS">Exit Fullscreen Mode</button>

<jh-script>
  <j-add-evt var="window" data-evtype="resize"
    data-add-evt>
     <j-fx param data-fx>
       <j-write element="'#ifCFS'" data-write-block>
          <j-isfullscreen method="active"/>
       </j-write>
     </j-fx>
   </j-add-evt>
   <j-add-evt element="#enterFS" data-elem-evtype="click"
     data-add-evt>
      <j-fx param data-fx>
        <j-fullscreen method="request"></j-fullscreen>
      </j-fx>
   </j-add-evt>
   <j-add-evt element="#exitFS" data-elem-evtype="click"
     data-add-evt>
      <j-fx param data-fx>
         <j-fullscreen method="exit"></j-fullscreen>
      </j-fx>
    </j-add-evt>
</jh-script>
```

- **Dates**
  - **Foundation**
    - Lets you deal with dates with different kinds of functions, for starters:

```
<j-date></j-date>
```

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
    <j-append element="'#date'" data-append-block>
        <j-date></j-date>
    </j-append>
</jh-script>
```

    - If you want to use date with **string** values:

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
    <j-append element="'#date'" data-append-block>
        <j-date data="'1999-02-23'" data-date></j-date>
    </j-append>
</jh-script>
```

    - If you want to use date with **just numbers**

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
    <j-append element="'#date'" data-append-block>
        <j-date data="1999, 1, 23" data-date></j-date>
    </j-append>
</jh-script>
```

- **Dates**
  - **Foundation**
    - For this kind of **entry**:

```
<j-date data="1999, 1, 23" data-date></j-date>
```

    - It returns February 23, 1999. The reason behind this is that JaihoeScript code is being translated to JavaScript, as JavaScript reads 0 as January up to 11 means December.

    - For only using **milliseconds**:

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
    <j-append element="'#date'" data-append-block>
        <j-date data="919699200000" data-date></j-date>
    </j-append>
</jh-script>
```

    - You could also do mathematical operation:

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
    <j-append element="'#date'" data-append-block>
        <j-date data="459849600000*2" data-date></j-date>
    </j-append>
</jh-script>
```

❖ Date Default Methods

➢ **To String** (Convert Date to String)

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
    <j-append element="'#date'" data-append-block>
       <j-date data="919699200000" data-date-to="string"
         data-date="extend"></j-date>
    </j-append>
</jh-script>
```

➢ **To Date String** (Convert to Date Only)

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
   <j-append element="'#date'" data-append-block>
     <j-date data="919699200000" data-date-to="DateString"
       data-date="extend"></j-date>
   </j-append>
</jh-script>
```

➢ **To UTC Date** (Convert to UTC Date Standard)

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
   <j-append element="'#date'" data-append-block>
       <j-date data="919699200000" data-date-to="UTCString"
        data-date="extend"></j-date>
   </j-append>
</jh-script>
```

❖ Date Default Methods

- **TO ISO Date** (Convert to ISO Date)

```
<j-date data="919699200000" data-date-to="ISOString"
  data-date="extend"></j-date>
```

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
   <j-append element="'#date'" data-append-block>
     <j-date data="919699200000" data-date-to="ISOString"
       data-date="extend"></j-date>
   </j-append>
</jh-script>
```

- **Date Parse** (Convert date to Milliseconds)

```
<j-date method="parse" data="'Feb 23, 1999'" data-date></j-date>
```

```
<p style="margin:0px;">
  Date (Feb 23, 1999) in Milliseconds:<span id="date"></span>
</p>
<jh-script>
 <j-append element="'#date'" data-append-block>
  <j-date method="parse" data="'Feb 23, 1999'" data-date></j-date>
 </j-append>
</jh-script>
```

- If you noticed, a method is used as it differs from the new date method

❖ Date Get Methods

➢ **Get Full Year** (Literally gets the year of the date)

```
<j-date data="'Feb 23, 1999'" data-date-get="year"
  data-date="extend"></j-date>
```

```
<p style="margin:0px;">The year is: <span id="date"></span></p>
<jh-script>
  <j-append element="'#date'" data-append-block>
    <j-date data="'Feb 23, 1999'" data-date-get="year"
      data-date="extend"></j-date>
  </j-append>
</jh-script>
```

➢ **Get Month** (Gets the month as a number staring from 0 to 11)

```
<j-date data="'Feb 23, 1999'" data-date-get="month"
  data-date="extend"></j-date>
```

```
<p style="margin:0px;">The month is: <span id="date"></span></p>
<jh-script>
  <j-let data="months" data-eq data-let>
    <j-date method="get" list="months"></j-date></j-let>
  <j-let data="monthKey" data-eq data-let>
    <j-date data="'Feb 23, 1999'" data-date-get="month"
      data-date="extend"></j-date></j-let>
  <j-append element="'#date'" data-append-block>
    <j-value data="months" data-key="monthKey" data-value="array">
      </j-value>
  </j-append>
</jh-script>
```

❖ Date Get Methods

➢ **Get Day** (Gets the day as a number from 0 to 6)

```
<j-date data="'Feb 23, 1999'" data-date-get="day"
  data-date="extend"></j-date>
```

```
<p style="margin:0px;">The day is: <span id="date"></span></p>
<jh-script>
 <j-let data="days" data-eq data-let>
   <j-date method="get" list="days"></j-date></j-let>
 <j-let data="dayKey" data-eq data-let>
   <j-date data="'Feb 23, 1999'" data-date-get="day"
     data-date="extend"></j-date></j-let>
 <j-append element="'#date'" data-append-block>
   <j-value data="days" data-key="dayKey" data-value="array">
 </j-value></j-append>
</jh-script>
```

➢ **Get Date** (Gets the date from 1 to 31)

```
<j-date data="'Feb 23, 1999'" data-date-get="date"
     data-date="extend"></j-date>
```

```
<p style="margin:0px;">The date from (Feb 23, 1999) is: <span
id="date"></span></p>
<jh-script>
 <j-append element="'#date'" data-append-block>
    <j-date data="'Feb 23, 1999'" data-date-get="date"
     data-date="extend"></j-date>
  </j-append>
</jh-script>
```

- **Date Get Methods**
  - **Get Hours** (Gets the date of hour from 0 to 23)

```
<j-date data data-date-get="hours" data-date="extend"></j-date>
```

  - **Get Minutes** (Gets the date of minutes from 0 to 59)

```
<j-date data data-date-get="minutes" data-date="extend"></j-date>
```

  - **Get Seconds** (Gets the date of seconds from 0 to 59)

```
<j-date data data-date-get="seconds" data-date="extend"></j-date>
```

  - **Get Milliseconds** (Gets the date of milliseconds from 0 to 999)

```
<j-date data data-date-get="msec" data-date="extend"></j-date>
```

  - Here is the **working code** of hours, minutes, seconds, milliseconds working **together**:

```
<p style="margin:0px;">(Time) <span id="time"></span></p>
<jh-script>
 <j-set-interval>
   <j-fx param data-fx>
     <j-write element="'#time'" data-write-block>
       <j-date data data-date-get="hours"
         data-date="extend"></j-date>
       <j-value data="+':'+" data-value />
       <j-date-lz><j-date data="" data-date-get="minutes"
         data-date="extend"></j-date></j-date-lz>
       <j-value data="+':'+" data-value />
       <j-date-lz><j-date data data-date-get="seconds"
         data-date="extend"></j-date></j-date-lz>
       <j-value data="+'.'+" data-value />
       <j-date data data-date-get="msec"
         data-date="extend"></j-date>
     </j-write>
   </j-fx>
   <j-interval time="10"></j-interval>
 </j-set-interval>
</jh-script>
```

  - **Wrap** minutes and seconds with this: (For leading zeros)
  - If you will use it as a clock if not, then don't

```
<j-date-lz></j-date-lz>
```

    - **For example:**

```
<j-date-lz><j-date data="" data-date-get="minutes/seconds"
  data-date="extend"></j-date></j-date-lz>
```

❖ Date Get Methods

➢ **Get Time** (A Method that gets the milliseconds starting from Jan 1, 1970)

```
<p style="margin:0px;">Date(getTime): <span id="time"></span></p>
<jh-script>
  <j-write element="'#time'" data-write-block>
    <j-date data="'Feb 23, 1999'" data-date-get="time"
      data-date="extend"></j-date>
  </j-write>
</jh-script>
```

➢ **Date Now** (Returns the milliseconds starting from Jan 1, 1970)

```
<p style="margin:0px;">Date(now): <span id="time"></span></p>
<jh-script>
  <j-write element="'#time'" data-write-block>
    <j-date-now />
  </j-write>
</jh-script>
```

➢ **Date Age** (Returns the age of the date, or the year to be specific)

```
<p style="margin:0px;">
 Date Age (Feb 23, 1999): <span id="age"></span>
</p>
<jh-script>
 <j-write element="'#age'" data-write-block>
  <j-date data-age="'Feb 23, 1999'" data-date></j-date>
 </j-write>
</jh-script>
```

- **Date Get Gap Methods**
  - These methods are exclusive only on Jaihoe
  - **Get Age Month** (Calculate how many months from now)

```
<j-date method="gap" type="month" data="dateSel" data-date></j-date>
```

```
<p style="margin:0px;">Date(2/23/1999) is: <span id="age"></span></p>
<jh-script>
  <j-write element="'#age'" data-write-block>
    <j-date method="gap" type="month" data="'Feb 23, 1999'"
      data-date></j-date>
    <j-value-add/><j-value data="' months from now'" data-value />
   </j-write>
</jh-script>
```

  - If the date input is in the future it will **return 0**

  - **Get Gap Day** (Calculate how many days from now)

```
<j-date method="gap" type="day" data="dateSel" data-date></j-date>
```

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
  <j-let data="dateSel = 'Jan 15, 2023'" data-let></j-let>
  <j-write element="'#date'" data-write-block>
    <j-date method="gap" type="day" data="dateSel" data-date></j-date>
    <j-value-add/><j-value data="' days to go now before '+dateSel"
      data-value></j-value>
   </j-write>
</jh-script>
```

  - If the date input is in the past, it will return a **negative number**, if you don't want that make an if statement then if it's a negative number then make it 0

- **UTC Get Methods**
  - Returns the UTC date of the selected method (Date)
    - **Get UTC Full Year**

```
<j-date data="'Feb 23, 1999'" data-dget-utc="year"
  data-date="extend"></j-date>
```

    - **Get UTC Month**

```
<j-date data="'Feb 23, 1999'" data-dget-utc="month"
  data-date="extend"></j-date>
```

    - **Get UTC Day**

```
<j-date data="'Feb 23, 1999'" data-dget-utc="day"
  data-date="extend"></j-date>
```

    - **Get UTC Date**

```
<j-date data="'Feb 23, 1999'" data-dget-utc="date"
  data-date="extend"></j-date>
```

- Here is the working code of the **preview**:

Using **UTC Methods**

```
<p style="margin:0px;">UTC Date (Fix): <span id="date"></span></p>
<jh-script>
  <j-let data="dateSel = 'Feb 23, 1999'" data-let></j-let>

  <j-let data="months" data-eq data-let>
    <j-date method="get" list="months"></j-date></j-let>
  <j-let data="monthKey" data-eq data-let>
    <j-date data="dateSel" data-dget-utc="month"
      data-date="extend"></j-date></j-let>
  <j-let data="days" data-eq data-let>
    <j-date method="get" list="days"></j-date></j-let>
  <j-let data="dayKey" data-eq data-let>
    <j-date data="dateSel" data-dget-utc="day"
      data-date="extend"></j-date></j-let>
  <j-let data="date" data-eq data-let>
    <j-date data="dateSel" data-dget-utc="date"
      data-date="extend"></j-date></j-let>
  <j-set data="date = date + 1" data-set></j-set>
  <j-write element="'#date'" data-write-block>
    <j-value data="days" data-key="dayKey+1" data-value="array"></j-value>
    <j-value data="+', '+" data-value></j-value>
    <j-value data="months" data-key="monthKey" data-value="array"></j-value>
    <j-value data="+' '+date+', '+" data-value></j-value>
    <j-date data="dateSel" data-dget-utc="year" data-date="extend"></j-date>
  </j-write>
</jh-script>
```

❖ UTC Get Methods

- Returns the UTC date of the selected method (Time)
  - **Get UTC Hours**

```
<j-date data data-dget-utc="hours" data-date="extend">
  </j-date>
```

  - **Get UTC Minutes**

```
<j-date data data-dget-utc="minutes" data-date="extend">
  </j-date>
```

  - **Get UTC Seconds**

```
<j-date data data-dget-utc="seconds" data-date="extend">
  </j-date>
```

  - **GET UTC Milliseconds**

```
<j-date data data-dget-utc="msec" data-date="extend">
  </j-date>
```

  - Here is the working code of using UTC Get Methods for time

```
<p style="margin:0px;">(UTC Time) <span id="time"></span></p>
<jh-script>
 <j-set-interval>
   <j-fx param data-fx>
     <j-write element="'#time'" data-write-block>
       <j-date data data-dget-utc="hours"
         data-date="extend"></j-date>
       <j-value data="+':'+" data-value />
       <j-date-lz><j-date data="" data-dget-utc="minutes"
         data-date="extend"></j-date></j-date-lz>
       <j-value data="+':'+" data-value />
       <j-date-lz><j-date data data-dget-utc="seconds"
         data-date="extend"></j-date></j-date-lz>
       <j-value data="+'.'+" data-value />
       <j-date data data-dget-utc="msec"
         data-date="extend"></j-date>
     </j-write>
   </j-fx>
   <j-interval time="10"></j-interval>
 </j-set-interval>
</jh-script>
```

❖ Date Set Methods

➢ **Set Full Year** (to set the year of the date)

```
<j-date-set data="d" data-set-date="year" param="1999"
  data-date="set-block"></j-data-set>
```

```
<p style="margin:0px;">Set Date: <span id="date"></span></p>
<jh-script>
  <j-let data="d" data-eq data-let><j-date data data-date>
    </j-date></j-let>
  <j-date-set data="d" data-set-date="year" param="1999"
    data-date="set-block"></j-data-set>
  <j-write element="'#date'" method="write" output="d"
    data-write></j-write>
</jh-script>
```

➢ **Set Month** (to set the month of the date)

```
<j-date-set data="d" data-set-date="month" param="1"
  data-date="set-block"></j-data-set>
```

```
<p style="margin:0px;">Set Date: <span id="date"></span></p>
<jh-script>
  <j-let data="d" data-eq data-let><j-date data data-date>
    </j-date></j-let>
  <j-date-set data="d" data-set-date="month" param="1"
    data-date="set-block"></j-data-set>
  <j-write element="'#date'" method="write" output="d"
    data-write></j-write>
</jh-script>
```

❖ Date Set Methods

- **Set Date** (to set the date)

```
<j-date-set data="d" data-set-date param="offsetDate"
  data-date="set-block"></j-data-set>
```

```
<p style="margin:0px;">Set Date (<span id="offset"></span>):
   <span id="date"></span>
</p>
<jh-script>
   <j-let data="d" data-eq data-let><j-date data data-date /></j-let>
   <j-let data="o" data-eq data-let><j-date method="offset"
     type="date" data="'Feb 23, 1999'" data-date /></j-let>
   <j-write element="'#offset'" method="write" output="o"
     data-write></j-write>
   <j-date-set data="d" data-set-date param="o"
     data-date="set-block"></j-data-set>

   <j-let data="nd" data-eq data-let><j-date data="d"
     data-date-to="DateString" data-date="extend"></j-date></j-let>
   <j-write element="'#date'" method="write" output="nd"
     data-write></j-write>
 </jh-script>
```

- To target a specific date, use **getOffsetDate** method, exclusive on Jaihoe

```
<j-date method="offset" type="date" data="'dateEntry'"
  data-date></j-date>
```

- **Set Hours** (to set the hour of the date)

```
<j-date-set data="h" data-set-date="hours" param="7"
  data-date="set-block"></j-data-set>
```

- **Set Minutes** (to set minutes of the date)

```
<j-date-set data="m" data-set-date="minutes" param="30"
  data-date="set-block"></j-data-set>
```

- **Set Seconds** (to set seconds of the date)

```
<j-date-set data="d" data-set-date="seconds" param="15"
  data-date="set-block"></j-data-set>
```

❖ Date Set Methods

➢ Here is the working code of using set hours, minutes and seconds:

```
<p style="margin:0px;">Set HMS: <span id="date"></span></p>
<jh-script>
  <j-let data="d" data-eq data-let><j-date data data-date>
    </j-date></j-let>
  <j-date-set data="d" data-set-date="hours" param="7"
    data-date="set-block"></j-data-set>
  <j-date-set data="d" data-set-date="minutes" param="30"
    data-date="set-block"></j-data-set>
  <j-date-set data="d" data-set-date="seconds" param="15"
    data-date="set-block"></j-data-set>
  <j-write element="'#date'" method="write" output="d"
    data-write></j-write>
</jh-script>
```

➢ **Set Milliseconds** (to set milliseconds of the date)

```
<j-date-set data="d" data-set-date="msec" param="125"
    data-date="set-block"></j-data-set>
```

```
<p style="margin:0px;">Set MSEC: <span id="date"></span></p>
<jh-script>
  <j-let data="d" data-eq data-let><j-date data data-date>
    </j-date></j-let>
  <j-date-set data="d" data-set-date="msec" param="125"
    data-date="set-block"></j-data-set>
  <j-let data="gd" data-eq data-let>
    <j-date-sg data="d" data-date-get="msec"
      data-date="extend"></j-date-sg>
  </j-let>
  <j-write element="'#date'" method="write" output="gd"
    data-write></j-write>
</jh-script>
```

➢ Use date set get (j-date-sg) to avoid redeclaring the variable d

```
<j-date-sg data="d" data-date-get="msec"
  data-date="extend"></j-date-sg>
```

- **Set Time** (to set time of the date)

```
<p style="margin:0px;">Set Time: <span id="date"></span></p>
<jh-script>
  <j-let data="d" data-eq data-let><j-date data data-date>
    </j-date></j-let>
  <j-date-set data="d" data-set-date="time"
    param="919699200000" data-date="set-block"></j-data-set>
  <j-write element="'#date'" method="write" output="d"
    data-write></j-write>
</jh-script>
```

❖ Locale Methods

- Enables you to use local conventions
  (Using the **New Date Method** or **Date Set Get Method**)
  - **Locale Date String** (gets the date portion of the date object as string)

```
<j-date-sg data="d" data-date-to="LocaleDateString"
  param="'en-PH',options" data-date="set-block"></j-date-sg>
```

```
<p style="margin:0px;">Date: <span id="date"></span></p>
<jh-script>
  <j-let data="d" data-eq data-let><j-date data data-date>
    </j-date></j-let>
  <j-var data="options" data-eq data-var><j-object>
    <j-attr name="weekday" data-value-sq="short" data-attr-sq />
    <j-attr name="year" data-value-sq="numeric" data-attr-sq />
    <j-attr name="month" data-value-sq="2-digit" data-attr-sq />
    <j-attr name="day" data-value-sq="numeric" data-attr-sq-close />
  </j-object></j-var>
  <j-let data="ld" data-eq data-let>
    <j-date-sg data="d" data-date-to="LocaleDateString"
      param="'en-PH', options" data-date></j-date-sg></j-let>
  <j-write element="'#date'" method="write" output="ld"
    data-write></j-write>
</jh-script>
```

❖ **Locale Methods**

➢ **Locale Time String** (gets the time portion of the date object as string)

```
<j-date-sg data="d" data-date-to="LocaleTimeString"
  param="[], {timeStyle : 'short'}" data-date></j-date-sg>
```

```
<p style="margin:0px;">Time: <span id="time"></span></p>
<jh-script>
  <j-let data="d" data-eq data-let><j-date data data-date>
    </j-date></j-let>
  <j-let data="lt" data-eq data-let>
    <j-date-sg data="d" data-date-to="LocaleTimeString"
      param="[], {timeStyle : 'short'}" data-date></j-date-sg></j-let>
  <j-write element="'#time'" method="write" output="lt"
    data-write></j-write>
</jh-script>
```

➢ **Locale String** (gets the string from the date object)

```
<j-date data data-date-to="LocaleString" param data-date></j-date>
```

```
<p style="margin:0px;">Time: <span id="time"></span></p>
<jh-script>
   <j-let data="ls" data-eq data-let>
     <j-date data data-date-to="LocaleString"
       param data-date></j-date></j-let>
   <j-write element="'#time'" method="write" output="ls"
     data-write></j-write>
</jh-script>
```

- **Redirect**
  - Redirect link into a specific location, window, depending on the target
    - **Link (Default)**
      - Redirect to a specified link and you can go back to the previous page

```
<j-redirect href="'https://google.com/'" data-redirect />
```

```
<p style="margin:0px;">Link: https://google.com/</p>
<button id="linkTest">Go to link</button>
<jh-script>
  <j-add-evt element="#linkTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-redirect href="'https://google.com/'" data-redirect />
     </j-fx>
  </j-add-evt>
</jh-script>
```

- **Replace**
  - Redirect to a specific link but you cant go to the previous page

```
<j-redirect link="replace" href="'http://google.com/'" data-redirect />
```

```
<p style="margin:0px;">Link Replace: http://google.com/</p>
<button id="linkTest">Go to link</button>
<jh-script>
  <j-add-evt element="#linkTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-redirect link="replace" href="'http://google.com/'"
         data-redirect />
     </j-fx>
  </j-add-evt>
</jh-script>
```

- **New Tab**
  - Redirect to a specific link into a new tab, or a window in old browsers

```
<j-redirect link="tab" href="'http://google.com/'" data-redirect />
```

```
<p style="margin:0px;">Link New Tab: http://google.com/</p>
<p style="margin:0px;">
  In some old browsers, it might open a new window.
</p>
<button id="linkTest">Go to link</button>
<jh-script>
  <j-add-evt element="#linkTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-redirect link="tab" href="'http://google.com/'"
         data-redirect />
     </j-fx>
  </j-add-evt>
</jh-script>
```

- **Window**
  - Redirect to a specific link into a window popup and its customizable
  - **Example 1**: Default Window

```
<j-redirect link="window" href="'http://google.com'" data-redirect />
```

❖ Window

- Here is the working code of the preview: (Example 1)

```
<p style="margin:0px;">Link Window: http://google.com</p>
<button id="linkTest">Open Window</button>
<jh-script>
  <j-add-evt element="#linkTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-redirect link="window" href="'http://google.com'"
         data-redirect />
     </j-fx>
  </j-add-evt>
</jh-script>
```

- **Example 2**: With Attributes
  - You can control the size of the window

```
<p style="margin:0px;">
  Link Window: https://jaihoejs.github.io/sw/
</p>
<button id="linkTest">Open Window</button>
<jh-script>
  <j-add-evt element="#linkTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-redirect link="window"
         href="'https://jaihoejs.github.io/sw/'"
         data-window-features="width:300,height:100"
         data-redirect="window" />
     </j-fx>
  </j-add-evt>
</jh-script>
```

- **Example 3**: Separate Attributes
  - You can separate the features of the window, it will produce a same result.

- **Example 3**: Separate Attributes
  - Here is the working code of the preview (feat for other features)

```
<p style="margin:0px;">Link Window: https://jaihoejs.github.io/sw/</p>
<button id="linkTest">Open Window</button>
<jh-script>
 <j-add-evt element="#linkTest" data-elem-evtype="click"
   data-add-evt>
   <j-fx param data-fx>
    <j-var data="features" data-eq data-var><j-object>
     <j-attr name="width" data-value-any="300" data-attr-any />
     <j-attr name="height" data-value-any="100" data-attr-any />
     <j-attr name="feat" data-value-sq="scrollbar=yes, resizable=yes"
      data-attr-sq-close />
    </j-object></j-var>
    <j-redirect link="window" href="'https://jaihoejs.github.io/sw/'"
      data-window-features="...features" data-redirect="window" />
   </j-fx>
 </j-add-evt>
</jh-script>
```

- **Download**
  - Check the specific link (within your files/server) and download it if it's a file
    - **Example 1**: Default Download

```
<j-redirect link="download" href="'./hello.txt'" data-redirect />
```

```
<p style="margin:0px;">Download Link: (./hello.txt)</p>
<button id="linkTest">Download Text File</button>
<jh-script>
  <j-add-evt element="#linkTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-redirect link="download" href="'./hello.txt'"
         data-redirect />
     </j-fx>
  </j-add-evt>
</jh-script>
```

    - Download **sample file** here: https://jaihoejs.github.io/dl/hello.txt then place the text file at the same location of this page

- **Example 2**: Default Download with file name

```
<j-redirect link="download" href="'./hello.txt'"
  data-download-filename="greet.txt" data-redirect="download" />
```

```
<p style="margin:0px;">Download Link: (./hello.txt > greet.txt)</p>
<button id="linkTest">Download Text File</button>
<jh-script>
  <j-add-evt element="#linkTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-redirect link="download" href="'./hello.txt'"
         data-download-filename="greet.txt"
         data-redirect="download" />
     </j-fx>
  </j-add-evt>
</jh-script>
```

  - If you noticed, the file name is renamed to greet.txt, this is best used if the link has no file extension into it

❖ Blob Download

- Fetch the specific link (within your files/server or other websites) and download it if it's a file using a blob link

```
<j-redirect link="download-blob"
  href="'https://jaihoejs.github.io/dl/hello.txt'" data-redirect />
```

❖ Blob Download

- **Example 1**: Here is the preview of a downloaded file from Blob

- If you noticed, the **url source** is changed to **Blob**, because:
  - The link was fetch and converted to Blob (object)
  - Then the Blob (object) is converted to an Object URL for it to be downloadable
  - That's why the link looks like gibberish or not readable
- If you will use this download method, make sure the file is not that big

- Here is the working code of the preview: (**Example 1**)

```
<p style="margin:0px;">
 Download Link: (https://jaihoejs.github.io/dl/hello.txt)
</p>
<button id="linkTest">Download Text File (Blob Link)</button>
<jh-script>
  <j-add-evt element="#linkTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-redirect link="download-blob"
         href="'https://jaihoejs.github.io/dl/hello.txt'"
         data-redirect />
     </j-fx>
  </j-add-evt>
</jh-script>
```

- **Blob Download**
  - **Example 2**: Here is the preview of a downloaded blob with different file name incase the url does not end with a file extension

```
<j-redirect link="download-blob" href="'https://picsum.photos/300/300'"
  data-download-filename="sample.jpeg" data-redirect="download" />
```

```
<p style="margin:0px;">
  Download Link: (https://picsum.photos/300/300)
</p>
<p style="margin:0 0 5px 0;">Image Name: (sample.jpeg)</p>
<button id="linkTest">Download Random Image (Blob Link)</button>
<jh-script>
  <j-add-evt element="#linkTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-redirect link="download-blob"
         href="'https://picsum.photos/300/300'"
         data-download-filename="sample.jpeg"
         data-redirect="download" />
     </j-fx>
  </j-add-evt>
</jh-script>
```

  - If you notice, every time you download an image it produces a different image and the url does not have a file extension at the end like .jpeg, .png or other image format. That's why it needs a **filename**.

- **IFrame**
  - Redirect to a specific link into an Iframe

```
<j-redirect link="iframe" href="'http://www.rdrop.com/'"
  data-iframe="'thisFrame'" data-redirect="iframe" />
```

```
<iframe src="http://www.taco.com/" name="thisFrame" height="300px"
width="480px" title="Iframe Example"></iframe>
<p style="margin:0px;"><b>Current Link:</b> http://www.taco.com/</p>
<p style="margin:0 0 5px 0;">
 <b style="color:red;">Open Link to iFrame:</b> http://www.rdrop.com/
</p>
<button id="linkTest">Open to iFrame</button>
<jh-script>
  <j-add-evt element="#linkTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-redirect link="iframe" href="'http://www.rdrop.com/'"
         data-iframe="'thisFrame'" data-redirect="iframe" />
     </j-fx>
  </j-add-evt>
</jh-script>
```

  - Make sure to indicate the name of the Iframe properly, otherwise it will open to a new tab or window

- **Window**
  - This represents the window of the browser, containing objects, functions and variables (even members) within a window
  - **Foundation**
    - Here is how you use **window** as a function (**Example #1**)

```
<j-window data="print" data-wcfx data-window="block"></j-window>
```

```
<p style="margin:0px;">Use window print: </p>
<button id="winTest">Print</button>
<jh-script>
 <j-add-evt element="#winTest" data-elem-evtype="click"
   data-add-evt>
  <j-fx param data-fx>
   <j-window data="print" data-wcfx data-window="block">></j-window>
  </j-fx>
 </j-add-evt>
</jh-script>
```

- **<j-window data="print"** is like window.print
- **data-wcfx** stands for window closed function "()"
- **data-window="block"** adds semicolon to the end, removing the block keyword removes it

- Here is how you use **window** as a variable (**Example #2**)

```
<j-window data="screen" data-inherit="width" data-window></j-window>
```

- **<j-window data="screen"** is like "window.screen"
- **data-inherit="width"** is like ".width"
- **data-window=""** does not add semicolon for variable use

- Here is the working example of the preview: (**Example #2**)

```
<p style="margin:0px;">Window Screen: <span id="wd"></span></p>
<jh-script>
  <j-let data="w" data-eq data-let>
    <j-window data="screen" data-inherit="width" data-window>
      </j-window>
  </j-let>
  <j-let data="h" data-eq data-let>
    <j-window data="screen" data-inherit="height" data-window>
      </j-window>
  </j-let>
  <j-write element="'#wd'" method="write" output="`${w}x${h}`"
    data-write></j-write>
</jh-script>
```

- **Window**
  - **Append**
    - If you just want to append **window** in your code to a function or any block (Example #3)

```
<j-window/><j-block data="print" data-wcfx data-block></j-block>
```

```
<p style="margin:0px;">Use window print: </p>
<button id="winTest">Print</button>
<jh-script>
 <j-add-evt element="#winTest" data-elem-evtype="click"
   data-add-evt>
  <j-fx param data-fx>
   <j-window/><j-block data="print" data-wcfx data-block></j-block>
  </j-fx>
 </j-add-evt>
</jh-script>
```

- **Window**
  - **Append**
    - Here is the working example of the preview (Example #3)

```
<p style="margin:0px;">Use window print: </p>
<button id="winTest">Print</button>
<jh-script>
 <j-add-evt element="#winTest" data-elem-evtype="click"
   data-add-evt>
  <j-fx param data-fx>
   <j-window/><j-block data="print" data-wcfx data-block></j-block>
  </j-fx>
 </j-add-evt>
</jh-script>
```

    - If you just want to append **window** in your code to a variable or any block (Example #3)

```
<j-window/><j-block data="screen" data-inherit="height"
  data-block="var"></j-block>
```

```
<p style="margin:0px;">Window Screen: <span id="wd"></span></p>
<jh-script>
  <j-let data="w" data-eq data-let>
    <j-window/><j-block data="screen" data-inherit="width"
      data-block="var"></j-block>
  </j-let>
  <j-let data="h" data-eq data-let>
    <j-window/><j-block data="screen" data-inherit="height"
      data-block="var"></j-block>
  </j-let>
  <j-write element="'#wd'" method="write" output="`${w}x${h}`"
    data-write></j-write>
</jh-script>
```

    - data-block="var" removes the semicolon for the block method

- **JSON (HACK)**
  - **Introduction**
    - JSON (JavaScript Object Notation) in **Jaihoe** is **HACK** which stands for **(HTML Applied Copy and Keep)**
    - It allows you to **keep** string, number, array, boolean, object variables into **string**
    - Then the **string** can be **copied** and parse it to an object to make it work in to the code
    - This allows you to **exchange data** between web servers, including programming languages that support **JSON**

  - **Parse (Copy)**
    - Copies a **string** object and **parse** it into a workable **variable** object
      - **Inline Method**
        - Lets you parse JSON and escape double or single quotes with the data-inline attribute to use JSON-HACK variant data

```
<j-hack data-inline data-copy="'{JSON-HACK variant data}'"
  data-hack></j-hack>
```

```
<p style="margin:0px;">Inline JSON in Jaihoe:</p>
<p id="binfo" style="margin:5px 0 0 0;">
  <b>Bakery Info:</b>
</p>
<jh-script>
  <j-let data="bakery" data-eq data-let>
    <j-hack data-inline
      data-copy="'{**name**:**John*s Bakery**, **age**:12}'"
      data-hack></j-hack>
    </j-let>
  <j-append element="'#binfo'" data-append-block>
    <j-value-rbr/><j-value data="'Name :'+bakery.name"
      data-value></j-value><j-value-br/>
    <j-value data="'Age: '+bakery.age" data-value></j-value>
  </j-append>
</jh-script>
```

- **JSON (HACK)**
  - **Parse (Copy)**
    - **Default Method**
      - You have to create a **JSON variable holder** or fetch it somewhere:

```
<j-let data="thisJSON" data-eq data-let>
  <j-valuearea hidden>
    {"name":"John's Bakery", "age":12}
  </j-valuearea>
</j-let>
```

      - Then **enclose** it with **this**:

```
<j-hack data-copy="thisJSON" data-hack></j-hack>
```

      - Here is the **working code**, it will still produce the same result:

```
<p style="margin:0px;">JSON in Jaihoe:</p>
<p id="binfo" style="margin:5px 0 0 0;"><b>Bakery Info:</b></p>
<jh-script>
  <j-let data="thisJSON" data-eq data-let>
    <j-valuearea hidden>
      {"name":"John's Bakery", "age":12}
    </j-valuearea>
  </j-let>
  <j-let data="bakery" data-eq data-let>
    <j-hack data-copy="thisJSON" data-hack></j-hack>
  </j-let>
  <j-append element="'#binfo'" data-append-block>
    <j-value-rbr/><j-value data="'Name :'+bakery.name"
      data-value></j-value><j-value-br/>
    <j-value data="'Age: '+bakery.age" data-value></j-value>
  </j-append>
</jh-script>
```

- **JSON (HACK)**
  - **Parse (Copy)**
    - **Block Method**
      - You can use copy block instead of separating the JSON in a variable

```
<p style="margin:0px;">JSON in Jaihoe:</p>
<p id="binfo" style="margin:5px 0 0 0;"><b>Bakery Info:</b></p>
<jh-script>
  <j-let data="bakery" data-eq data-let>
    <j-hack-block data-copy>
      <j-valuearea hidden>
        {"name":"John's Bakery", "age":12}
      </j-valuearea>
    </j-hack-block>
  </j-let>
    <j-append element="'#binfo'" data-append-block>
      <j-value-rbr/><j-value data="'Name :'+bakery.name"
        data-value></j-value><j-value-br/>
      <j-value data="'Age: '+bakery.age" data-value></j-value>
    </j-append>
</jh-script>
```

      - Also works code it like this:

```
<p style="margin:0px;">JSON in Jaihoe:</p>
<p id="binfo" style="margin:5px 0 0 0;"><b>Bakery Info:</b></p>
<jh-script>
  <j-let data="bakeryData" data-eq data-let>
    <j-valuearea hidden>{"name":"John's Bakery", "age":12}</j-valuearea>
  </j-let>
  <j-let data="bakery" data-eq data-let><j-hack-block data-copy>
    <j-value data="bakeryData" data-value></j-value></j-hack-block>
  </j-let>
  <j-append element="'#binfo'" data-append-block>
    <j-value-rbr/><j-value data="'Name :'+bakery.name"
      data-value></j-value><j-value-br/>
    <j-value data="'Age: '+bakery.age" data-value></j-value>
  </j-append>

</jh-script>
```

- **Stringify (Keep)**
  - Converts a workable **variable** object to a **string** object which can be **copied** later
  - In other terms, it converts the **variable** object to string for **keeps** and can be retrieved and **copied** for later use

- **Default Method**

```
<j-hack data-keep="object/variable" data-hack></j-hack>
```

```
<p style="margin:0px;">JSON in Jaihoe:</p>
<p id="binfo" style="margin:5px 0 0 0;"><b>Bakery Info:</b></p>

<jh-script>
  <j-var data="bakeryInfo" data-eq data-var>
    <j-object>
      <j-attr name="name" value="John's Bakery" data-attr></j-attr>
      <j-attr name="age" data-value-any="12" data-attr-any-close>
    </j-object>
  </j-var>
  <j-var data="thisJSON" data-eq data-var>
    <j-hack data-keep="bakeryInfo" data-hack></j-hack>
  </j-var>
  <j-append element="'#binfo'" data-append-block>
    <j-value-rbr/><j-value data="'JSON (String): '+thisJSON"
      data-value></j-value>
  </j-append>
  <j-var data="bakery" data-eq data-var>
   <j-hack data-copy="thisJSON" data-hack></j-hack>
  </j-var>
  <j-append element="'#binfo'" data-append-block>
    <j-value-rbr/><j-value data="'Name :'+bakery.name"
      data-value></j-value><j-value-br/>
    <j-value data="'Age: '+bakery.age" data-value></j-value>
  </j-append>
</jh-script>
```

- **Stringify (Keep)**
  - **Block Method**
    - You can use keep block instead of separating the object variable

```
<j-hack-block data-keep>
  <j-object>
    <j-attr name="name" value="John's Bakery" data-attr></j-attr>
    <j-attr name="age" data-value-any="12" data-attr-any-close>
  </j-object>
</j-hack-block>
```

```
<p style="margin:0px;">JSON in Jaihoe:</p>
<p id="binfo" style="margin:5px 0 0 0;"><b>Bakery Info:</b></p>

<jh-script>
   <j-let data="bakery" data-eq data-let>
     <j-hack-block data-copy>
       <j-hack-block data-keep>
         <j-object>
          <j-attr name="name" value="John's Bakery" data-attr></j-attr>
          <j-attr name="age" data-value-any="12" data-attr-any-close>
         </j-object>
       </j-hack-block>
     </j-hack-block>
   </j-let>
   <j-append element="'#binfo'" data-append-block>
     <j-value-rbr/><j-value data="'Name :'+bakery.name"
       data-value></j-value><j-value-br/>
     <j-value data="'Age: '+bakery.age" data-value></j-value>
   </j-append>
</jh-script>
```

- **WebStorage**
  - **Session Storage**
    - Allows you to store data temporarily (until the window is closed) using keys
      - **Set**
        - Stores a data within its key

```
<j-storage type="session" method="set" data-storage-key="'name'"
  data-storage-value="'Jerwin'" data-storage="block" />
```

      - **Get**
        - Gets a data that has been set with the key

```
<j-storage type="session" method="get" data-storage-key="'name'"
   data-storage />
```

      - Here is a working code of using **set** and **get** in Session Storage:

```
<p style="margin:0px;">Session Storage: <span id="sdata"></span></p>
<button id="updSS" type="button">Get Data</button>
<jh-script>
  <j-storage type="session" method="set" data-storage-key="'name'"
    data-storage-value="'Jerwin'" data-storage="block" />

 <j-add-evt element="#updSS" data-elem-evtype="click"
  data-add-evt>
   <j-fx param data-fx>
    <j-write element="'#sdata'" data-write-block>
     <j-storage type="session" method="get" data-storage-key="'name'"
       data-storage />
    </j-write>
   </j-fx>
  </j-add-evt>
</jh-script>
```

- **WebStorage**
  - **Session Storage**
    - **Remove**
      - Removes a data within its key

```
<j-storage type="session" method="remove" data-storage-key="'name'"
  data-storage-value="'23'" data-storage="block" />
```

      - Working code is on the next page, as it removes only the name.

    - **Clear**
      - Removes all data within the session storage

```
<j-storage type="session" method="clear" data-storage="block" />
```

      - Working code is on the next page, as it removes all the data.

➢ Here is the working code of using **remove** and **clear** in **session storage**:

```
<p style="margin:0px;"><b>Session Storage:</b> </p>
<p style="margin:5px 0 5px 0;" id="sdata"></p>
<button id="updSS" type="button">Get Data</button>
<button id="remSS" type="button">Remove Name</button>
<button id="clrSS" type="button">Clear Data</button>
<jh-script>
  <j-storage type="session" method="set" data-storage-key="'name'"
    data-storage-value="'John'" data-storage="block" />
  <j-storage type="session" method="set" data-storage-key="'age'"
    data-storage-value="'23'" data-storage="block" />

  <j-add-evt element="#updSS" data-elem-evtype="click"
     data-add-evt>
      <j-fx param data-fx>
        <j-write element="'#sdata'" data-write-block>
      <j-value data="'Name: '" data-value /><j-value-add/>
      <j-storage type="session" method="get" data-storage-key="'name'"
          data-storage /><j-value-br/>
          <j-value data="'Age: '" data-value /><j-value-add/>
          <j-storage type="session" method="get" data-storage-key="'age'"
            data-storage />
    </j-write>
      </j-fx>
   </j-add-evt>
   <j-add-evt element="#remSS" data-elem-evtype="click"
     data-add-evt>
      <j-fx param data-fx>
        <j-storage type="session" method="remove" data-storage-key="'name'"
        data-storage-value="'23'" data-storage="block" />
      </j-fx>
   </j-add-evt>
   <j-add-evt element="#clrSS" data-elem-evtype="click"
     data-add-evt>
      <j-fx param data-fx>
        <j-storage type="session" method="clear" data-storage="block" />
      </j-fx>
   </j-add-evt>
</jh-script>
```

- **WebStorage**
  - **Local Storage**
    - Allows you to store data locally on your browser using keys, unlike session storage the data is temporarily saved (until the window/tab is closed)
      - **Set**
        - Also stores data within its key and

```
<j-storage type="local" method="set" data-storage-key="'name'"
  data-storage-value="'Jack'" data-storage="block" />
```

      - **Get**
        - Gets a data that has been set with the key

```
<j-storage type="local" method="get" data-storage-key="'name'"
  data-storage />
```

        - Here is a working code of using **set** and **get** in Local Storage:

```
<p style="margin:0px;">Local Storage: <span id="sdata"></span></p>
<button id="updSS" type="button">Get Data</button>
<jh-script>
  <j-storage type="local" method="set" data-storage-key="'name'"
    data-storage-value="'Jack'" data-storage="block" />

 <j-add-evt element="#updSS" data-elem-evtype="click"
  data-add-evt>
   <j-fx param data-fx>
    <j-write element="'#sdata'" data-write-block>
     <j-storage type="local" method="get" data-storage-key="'name'"
       data-storage />
    </j-write>
   </j-fx>
  </j-add-evt>
</jh-script>
```

- **WebStorage**
  - **Local Storage**
    - **Remove**
      - Removes a data within its key

```
<j-storage type="local" method="remove" data-storage-key="'name'"
  data-storage-value="'23'" data-storage="block" />
```

    - **Clear**
      - Removes all data within the local storage

```
<j-storage type="local" method="clear" data-storage="block" />
```

➢ Here is the working code of using **remove** and **clear** in **local storage**:

```
<p style="margin:0px;"><b>Local Storage:</b> </p>
<p style="margin:5px 0 5px 0;" id="sdata"></p>
<button id="updSS" type="button">Get Data</button>
<button id="remSS" type="button">Remove Name</button>
<button id="clrSS" type="button">Clear Data</button>
<jh-script>
  <j-storage type="local" method="set" data-storage-key="'name'"
    data-storage-value="'Jill'" data-storage="block" />
  <j-storage type="local" method="set" data-storage-key="'age'"
    data-storage-value="'23'" data-storage="block" />

  <j-add-evt element="#updSS" data-elem-evtype="click"
     data-add-evt>
      <j-fx param data-fx>
        <j-write element="'#sdata'" data-write-block>
      <j-value data="'Name: '" data-value /><j-value-add/>
      <j-storage type="local" method="get" data-storage-key="'name'"
          data-storage /><j-value-br/>
          <j-value data="'Age: '" data-value /><j-value-add/>
          <j-storage type="local" method="get" data-storage-key="'age'"
            data-storage />
    </j-write>
      </j-fx>
   </j-add-evt>
   <j-add-evt element="#remSS" data-elem-evtype="click"
     data-add-evt>
      <j-fx param data-fx>
        <j-storage type="local" method="remove" data-storage-key="'name'"
        data-storage-value="'23'" data-storage="block" />
      </j-fx>
   </j-add-evt>
   <j-add-evt element="#clrSS" data-elem-evtype="click"
     data-add-evt>
      <j-fx param data-fx>
        <j-storage type="local" method="clear" data-storage="block" />
      </j-fx>
   </j-add-evt>
</jh-script>
```

- **Debugging**
  - **Window Console**
    - **Log (Default)**
      - Displays log information in the console area

```
<j-console param="'Hello world'" data-console></j-console>
```

```
<div id="chElem"></div>
<jh-script>
  <j-chelp element="#chElem" data-chelp-element></j-chelp>
  <j-console param="'Hello world'" data-console></j-console>
</jh-script>
```

    - **Clear**
      - Clears the console area

```
<j-console type="clear"></j-console>
```

```
<div id="chElem"></div>
<jh-script>
  <j-chelp element="#chElem" data-chelp-element></j-chelp>
  <j-console type="clear"></j-console>
</jh-script>
```

- **Assert**
  - Displays an assertion error if condition is false

```
<j-console type="assert" param="x+y == 10,'Not equal to 10'"
  data-console></j-console>
```

```
<div id="chElem"></div>
<jh-script>
 <j-chelp element="#chElem" data-chelp-element></j-chelp>
 <!-- Try to change x and y to 5, assert does not display -->
    <j-let data="x = 7" data-let></j-let>
    <j-let data="y = 7" data-let></j-let>
 <j-console type="assert" param="x+y == 10,'Not equal to 10'"
   data-console></j-console>
</jh-script>
```

- **Count**
  - Counts the number this method is called depending on its label or not

```
<j-console type="count" param="'forLoop'"
  data-console></j-console>
```

```
<div id="chElem"></div>
<jh-script>
 <j-chelp element="#chElem" data-chelp-element></j-chelp>
 <j-for start="let i = 0" scope="i < 5" action="i++" data-for>
   <j-console type="count" param="'forLoop'"
     data-console></j-console>
 </j-for>
</jh-script>
```

- **Error**
  - Displays an error in the console area

```
<j-console type="error" param="'An error has occurred.'"
  data-console></j-console>
```

```
<div id="chElem"></div>
<jh-script>
  <j-chelp element="#chElem" data-chelp-element></j-chelp>
  <j-console type="error" param="'An error has occurred.'"
    data-console></j-console>
</jh-script>
```

- **Warn**
  - Displays a warning in the console area

```
<j-console type="warn" param="'This is a warning!'" data-console>
  </j-console>
```

```
<div id="chElem"></div>
<jh-script>
 <j-chelp element="#chElem" data-chelp-element></j-chelp>
 <j-console type="warn" param="'This is a warning!'" data-console>
   </j-console>
</jh-script>
```

- **Info**
  - Similar to console log, it displays general information in the console area

```
<j-console type="info" param="'Hello World!'" data-console></j-console>
```

```
<div id="chElem"></div>
<jh-script>
 <j-chelp element="#chElem" data-chelp-element></j-chelp>
 <j-console type="info" param="'Hello World!'" data-console>
   </j-console>
</jh-script>
```

- **Group**
  - Defines a group within the other console types below

```
<j-console type="group" param data-console></j-console>
```

```
<div id="chElem"></div>
<jh-script>
 <j-chelp element="#chElem" data-chelp-element></j-chelp>
 <j-console param="'Hello World!'" data-console></j-console>
 <j-console type="group" param data-console></j-console>
 <j-console param="'Hello again, in a group!'" data-console>
   </j-console>
</jh-script>
```

- **Group Collapsed**
  - Defines a group collapsed within the other console types below

```
<j-console type="groupCollapsed" param data-console></j-console>
```

```
<div id="chElem"></div>
<jh-script>
  <j-chelp element="#chElem" data-chelp-element></j-chelp>
  <j-console param="'Hello World!'" data-console></j-console>
  <j-console type="groupCollapsed" param data-console></j-console>
  <j-console param="'Hello again, in a group collapse!'"
    data-console></j-console>
</jh-script>
```

- **Group End**
  - Defines a group end to close group or group collapsed above it

```
<j-console type="groupEnd"></j-console>
```

```
<div id="chElem"></div>
<jh-script>
 <j-chelp element="#chElem" data-chelp-element></j-chelp>
 <j-console param="'Hello World!'" data-console></j-console>
 <!-- You can replace group with groupCollapsed -->
 <j-console type="group" param data-console></j-console>
 <j-console param="'Hello again, in a group!'" data-console></j-console>
 <j-console type="groupEnd"></j-console>
 <j-console param="'And out of the group!'" data-console></j-console>
</jh-script>
```

- **Time**
  - Starts a timer in the console area

```
<j-console type="time" param="'forLoop'" data-console></j-console>
```

- **Time End**
  - Stops a timer in the console area

```
<j-console type="timeEnd" param="'forLoop'" data-console></j-console>
```

- Here is the working code of using **time** and **time end** in Jaihoe: (Example #1)

```
<div id="chElem"></div>
<jh-script>
 <j-chelp element="#chElem" data-chelp-element></j-chelp>
   <j-console type="time" param="'forLoop'" data-console></j-console>
     <j-for start="let i = 0" scope="i < 123456" action="i++" data-for>
     </j-for>
   <j-console type="timeEnd" param="'forLoop'" data-console></j-console>
</jh-script>
```

- **Trace**
  - Displays how the code ended up with the code in the console area

- **Trace**
  - Here is the working code of the preview:

```
<div id="chElem"></div>
<button style="margin-top:5px;" id="traceTest">Start Trace</button>
<jh-script>
  <j-chelp element="#chElem" data-chelp-element></j-chelp>
  <j-add-evt element="#traceTest" data-elem-evtype="click"
    data-add-evt>
     <j-fx param data-fx>
       <j-fx-call name="otherFx" param data-fx-call></j-fx-call>
     </j-fx>
  </j-add-evt>
  <j-fx name="otherFx" param data-fx>
    <j-console type="trace" param data-console></j-console>
  </j-fx>
</jh-script>
```

- **Table**
  - Displays a table in the console area

```
<j-console type="table" param="fruitCount" data-console></j-console>
```

```
<div id="chElem"></div>
<jh-script>
 <j-chelp element="#chElem" data-chelp-element></j-chelp>
  <j-let data="fruits" data-litarr data-let></j-let>
   <j-arrayx data="fruits" data-arrayx>
    <j-arelx param="'Banana',5" data-arelx></j-arelx>
    <j-arelx param="'Apple',1" data-arelx></j-arelx>
    <j-arelx param="'Orange','3'" data-arelx></j-arelx>
    <j-arelx param="'Mango'" data-arelx></j-arelx>
  </j-arrayx>
  <j-let data="fruitCount" data-eq data-let>
    <j-array-val method="jaihoe" type="eachArrNode"
      param="fruits" data-array-method></j-array-val>
  </j-let>
  <j-console param="fruits" data-console></j-console>
  <j-console type="table" param="fruitCount" data-console></j-console>
</jh-script>
```

- **Code View**
  - This is how you see what's going on within your JaihoeScript code
    - **Input Code (JaihoeScript)**
      - Displays the JaihoeScript Code regardless of how its written, it will still be formatted to be easily converted in JavaScript

```
<!-- ::Jaihoe::(Show JaihoeScript Code) -->
```

```
<p id="test"></p>
<jh-script>
  <!-- ::Jaihoe::(Show JaihoeScript Code) -->
  <j-write element="'#test'" method="write"
    output="'Hello World'" data-write>
  </j-write>
</jh-script>
```

    - **Executable Code (JavaScript)**
      - Displays the converted JavaScript code to be executed from the JaihoeScript code written that has been formatted

```
<!-- ::Jaihoe::(Show JavaScript Code) -->
```

```
<p id="test"></p>
<jh-script>
  <!-- ::Jaihoe::(Show JavaScript Code) -->
  <j-write element="'#test'" method="write"
    output="'Hello World'" data-write>
  </j-write>
</jh-script>
```

❖ Code View

➢ Library List (Jaihoe LibraryList)

➢ Displays the list of library used in array

```
<!-- ::Jaihoe::(Show Jaihoe LibraryList) -->
```

```
<jh-dep src="dependency.html">
  <!-- ::Jaihoe::(Show Jaihoe LibraryList) -->
  <j-imp name="jaihoe_currency" src="../libraries"></j-imp>
  <j-imp name="jaihoe_convert" src="../libraries"></j-imp>
</jh-dep>
```

➢ Library Paths (Jaihoe LibraryPaths)

➢ Displays the list of library used in array

```
<!-- ::Jaihoe::(Show Jaihoe LibraryPath) -->
```

```
<jh-dep src="dependency.html">
  <!-- ::Jaihoe::(Show Jaihoe LibraryPath) -->
  <j-imp name="jaihoe_currency" src="../libraries"></j-imp>
  <j-imp name="jaihoe_convert" src="../libraries"></j-imp>
</jh-dep>
```

- **Code Instants**
  - These are certain parts that you can add to your code if needed
    - **Attribute Lists**
      - Here are the are list of attribute instants

Name	Attribute	Equals	Usage Example	Output
Equals	data-eq	=	data-eq="value"	= value
Double Equals	data-eqd	==	data="num" data-eqd="5"	num == 5
Triple Equals	datq-eqt	===	data="name" data-eqt="'John'"	name === 'John'
Equals to Empty Array (Literal)	data-littar	= []	data="array" data-littar	array = []
Empty Array (Literal)	data-literal-array	[]	data="array" data-eq data-literal-array	array = []
Increment	data-inc	++	data="i" data-inc	i++
Decrement	data-dec	--	data="i" data-dec	i--
Addition Assignment	data-add-eq	+=	data="str" data-add-eq="'John'"	str += 'John'
Substraction Assignment	data-minus-eq	-=	data="10" data-minus-eq="5"	10 -= 5
Inherit	data-inherit	.	data="screen" data-inherit="width"	screen.width
Line break	data-br	" "	data="str" data-add data-br	str + " "
Put Empty	data-put	no value	data data-put="'Hi'"	'Hi'
Add	data-add	+	data="'Hello '" data-add data-put="'World'"	'Hello'+ 'World'
Substraction	data-minus	-	data="15" data-minus data-put="5"	15 - 5
This	data-this	this.	data data-this="age" data-eq="23"	this.age = 23
Parameter Start	param	(	<j-fx name="callFx" param data-fx>	function callFx(){
Parameter End	data-parame	)	data data-put="setAge" param data-parame	setAge()
Closed Parameter	data-cparam	()	data data-put="getAge" data-cparam	getAge()
New	data-new	new	data data-new="Age" param="23" data-parame	new Age(23)
Return	data-ret	return	data data-ret="'Hi'"	return 'Hi'
To JSON	data-json	.json()	data data-ret="data" data-json	return data.json()

- **Code Instants**
  - **Block Lists**

Name	Block	Equals	Usage Example	Output
Value Line Break	<j-value-obr/>	" "	<j-value data="'str'" data-add data-value/><j-value-obr>	'str' + " "
Add Line Break from the Left	<j-value-lbr/>	+" "	<j-value data="'str'" data-value/><j-value-lbr/>	'str' + " "
Add Line Break from the Right	<j-value-rbr/>	" "+	<j-value-rbr/><j-value data="'str'" data-value/>	" " + 'str'
Add Line Break in a middle of a text	<j-value-br/>	+" " +	<j-value data="'Hello'" data-value /> <j-value-br/> <j-value data="'World'" data-value />	'Hello' +" "+ 'World'
Value True	<j-value-true/>	true	<j-value data="old" data-eqd data-value/><j-value-true/>	old == true
Value False	<j-value-false/>	false	<j-value data="old" data-eqd data-value/><j-value-false/>	old == false
Value Addition	<j-value-add/>	+	<j-value data="'Hello '" data-value /> <j-value-add/> <j-value data="'World'" data-value />	'Hello' + 'World'
Value Subtraction	<j-value-minus/>	-	<j-value data="15" data-value /><j-value-minus/> <j-value data="5" data-value />	15 - 5
And	<j-and/>	,	<j-then-close data="promised" data-then> <j-fx param="value" data-fx></j-fx> <j-and/> <j-fx param="err" data-fx></j-fx> </j-then-close>	promised.then(function(value) {}, function(err) {});

- **Custom**
  - If there is a method that is missing, you can just put JavaScript methods inside of it

```
<j-custom-code hidden>
  <!-- INSERT CODE HRE -->
</j-custom-code>
```

```
<p id="chElem"></p>
<jh-script>
    <!-- ::Jaihoe::(Show JavaScript Code) -->
    <j-chelp element="#chElem" data-chelp-element></j-chelp>
    <j-custom-code hidden>
        document.body.style.backgroundColor = "pink";
        window.open();
    </j-custom-code>
</jh-script>
```

  - **Rules**:
    - You **cannot use** the following inside the **custom code block**:
      - eval, alert, close, confirm, open, print, prompt and window.
        - If case you use these **words**, it will turn as a **comment**

- **Bad Practice**
  - **Avoid** using HTML code inside attributes (It will not work)

```
<j-write element="'#test'" method="write"
  output='<j-value data="`Hello`" data-value></j-value>' data-write>
</j-write>
```

  - **Instead**, separate it in a variable

```
<j-let data="greet" data-eq="'Hello World'" data-let></j-let>
<j-write element="'#test'" method="write"
  output="greet" data-write></j-write>
```

Libraries

- **Convert**
  - **Foundation**
    - Specialized for converting units (experimental) with 12 variations including the following:

volume for Volume conversion	**area** for Area conversion
length for Length conversion	**speed** for Speed conversion
weight for Weight and Mass conversion	**time** for Time conversion
temp for Temperature conversion	**power** for Power conversion
energy for Energy conversion	**pressure** for Pressure conversion
data for Data conversion	**angle** for Angle conversion

- **Dependency Setup**: (Include the following in the HTML body)

```
<jh-dep>
    <!-- IMPORT DEPENDENCY -->
    <j-imp name="jaihoe_convert" src="./libraries"></j-imp>
</jh-dep>
```

- **Script Setup**: (Then include the following below the dependency block intended for convert *)

```
<p id="samplex"></p>
<jh-script>
  <j-var data="x" data-eq data-var>
    <j-convert>
       <!-- INSERT PROPERTIES HERE -->
    </j-convert>
  </j-var>
  <j-write element="'#samplex'" method="write" output="x"
    data-write></j-write>
</jh-script>
```

- Here is a preview of using **Convert**:

  - 3 Milliliters is equals to 0.003 Liters

❖ Working Code

➢ Here is the **working code** of the preview: (**Convert ***)

```
<jh-dep>
  <j-imp name="jaihoe_convert" src="./libraries"></j-imp>
</jh-dep>

<p id="samplex"></p>
<jh-script>
  <!-- Declare x variable to convert Milliliters to Liters -->
  <j-var data="x" data-eq data-var>
    <j-convert>
      <j-attr name="for" value="convert volume" data-attr></j-attr>
      <j-attr name="unitValue" value="3" data-attr></j-attr>
      <j-attr name="type" value="ml" data-attr></j-attr>
      <j-attr name="target" value="l" data-attr></j-attr>
      <j-attr name="forElement" value="#samplex" data-attr></j-attr>
      <j-attr name="tip" value="true" data-attr></j-attr>
      <j-attr name="separate" value="true" data-attr></j-attr>
      <j-attr name="unit" value="true" data-attr></j-attr>
      <j-attr name="display" value="all" data-attr></j-attr>
      <j-attr name="round" value="" data-close></j-attr>
    </j-convert>
  </j-var>
  <j-write element="'#samplex'" method="write" output="x"
    data-write></j-write>
</jh-script>
```

❖ Converting List

➢ If you don't know what to put on the **type** or **target**, you can simply enter the following: convert <name of the conversion method> list:

➢ For example: **Angle Conversion**

- **Converting List**
  - Here is the working code of using conversion list

```
<div id="conList"></div>
<jh-script>
  <j-convert>
    <j-attr name="for" value="convert angle list" data-attr></j-attr>
    <j-attr name="target" value="#conList" data-attr></j-attr>
  </j-convert>
</jh-script>
```

  - You can change the **angle** with the following below:
    - volume, area, length, speed, weight, time, temp, power, energy, pressure, data and angle

- **Required Attribute Modifiers**
  - **For Conversion** – Defines the conversion method

```
<j-attr name="for" value="convert volume" data-attr></j-attr>
```

  - **Conversion Unit Value** – This is the current conversion value to be converted

```
<j-attr name="unitValue" value="2" data-attr></j-attr>
```

  - **Conversion Type** – This is the current conversion type to be converted

```
<j-attr name="type" value="ml" data-attr></j-attr>
```

  - **Conversion Target** – This is the target conversion from the defined type conversion

```
<j-attr name="target" value="l" data-attr></j-attr>
```

    - For the values of conversion type and target, go to the foundation page or to the further page

- **Basic Attribute Modifiers**
  - **display** (all/converted) – **Display** style of the converted unit.

```
<j-attr name="display" value="all" data-attr></j-attr>
```

    - For Example: (Default is **converted**)
      - **1 kilograms converting to grams**
      - If **true**, it will show "1kg = 1000 g"; (unit is true)
      - If **false**, it will show "1000 g"; (unit is true)

  - **unit** (true/false) – **Unit** is simply for adding units to the converted unit.

```
<j-attr name="unit" value="true" data-attr></j-attr>
```

    - For example: (Default is **false**)
      - **2 kilograms converting to grams**
      - If **true**, it will show "2 kg = 2000 g" (display is all)
      - If **false**, it will show "2 = 2000" (display is all)
      - If **false**, it will show "2000" (display is converted)

  - **separate** (true/false) – Separates the whole number only not the decimal number if true, to exclude the feature don't include it on the block.

```
<j-attr name="separate" value="true" data-attr></j-attr>
```

    - For example: (Default is **false**)
      - **3 kilograms to converting to grams**
      - If **true**, it will show "3,000 g" (display is converted & unit is true)
      - If **false**, it will show "3000 g" (display is converted & unit is true)

  - **tip** (true/false) – **Additional accessory** to show what unit was used:

```
<j-attr name="tip" value="true" data-attr></j-attr>
<j-attr name="forElement" value="element" data-attr></j-attr>
```

    - For example: (Default is **false**)

- **tip** (true/false) – **Additional accessory**
  - **Take Note:** make sure to specify the **forElement** to the right element target if not it will target the script holder.
    - 1 kg = 1000 g || 1 = 1000 (display is All & with or without unit)
    - If **true**, it will show "1 kilograms = 1000 grams" if you hover it
    - 1kg (display is converted and unit is true)
    - If **true**, it will show "1 kilograms"
    - 1 (which is kg but unit is false and display is converted)
    - if **true**, it will show "kilograms"

- **strict** (true/false) – **Strict** if set to true it has a **bypass feature**:

```
<j-attr name="strict" value="false" data-attr></j-attr>
```

  - Ignores **non-numerical numbers** and returns the **numerical number** to make it **valid**.
    - For Example: (Default is **true**)
      - Like "Jerwin's 1000 kg of gold." **turns to** "1000" which is **valid**.
      - While "Jerwin's 1000 kg of gold." is **invalid** if set to false.
  - Including a negative number is possible for non-negative conversions.
    - For Example: Like "-1kg = -1000g" is **valid** while if false its **invalid**.
  - Same conversion is possible.
    - For Example: Like "1kg = 1kg" is **valid** while if false its **invalid**.

❖ Rounding Attribute Modifiers

- This is where the converted value can be rounded with the rounding methods:
  - **Round** (number input) - rounds the converted unit's decimal number

```
<j-attr name="round" value="6" data-close></j-attr>
```

    - For example: 1 fluid ounces (US) converting to gallons (US)
      - If **round**="6" then the converted unit is equal to **0.007813 gal (US)**
      - It you notice, it **rounds** the number.

- Here is the working code of using **Round**:

```
<p id="samplex"></p>
<jh-script>
  <j-var data="x" data-eq data-var>
    <j-convert>
      <j-attr name="for" value="convert volume" data-attr></j-attr>
      <j-attr name="unitValue" value="1" data-attr></j-attr>
      <j-attr name="type" value="fluid ounces us" data-attr></j-attr>
      <j-attr name="target" value="gallons us" data-attr></j-attr>
      <j-attr name="forElement" value="#samplex" data-attr></j-attr>
      <j-attr name="tip" value="true" data-attr></j-attr>
      <j-attr name="separate" value="true" data-attr></j-attr>
      <j-attr name="unit" value="true" data-attr></j-attr>
      <j-attr name="display" value="all" data-attr></j-attr>
      <j-attr name="round" value="6" data-close></j-attr>
    </j-convert>
  </j-var>
  <j-write element="'#samplex'" method="write" output="x"
    data-write></j-write>
</jh-script>
```

- **trunc** (number input) – Split the converted unit's decimal number

```
<j-attr name="trunc" value="6" data-close></j-attr>
```

  - For example: 1 fluid ounces (US) converting to gallons (US)
    - If **trunc** value is "6" then the converted unit is equal to **0.007812 gal (US)**
    - If you notice, It **splits** the raw converted unit & not rounded.

- For the **working code** use the code of **Round** and replace the **round attribute** to **trunc**.

- **Exponential Attribute Modifiers**
  - This is where you deal with 'e' notation and here are the methods:
    - **wn-auto** (true/false) – **Whole Number Auto** if set to **true** it extracts the converted unit in exponential form to a whole number (or its in decimal form) if possible. Not all converted units are applicable.

```
<j-attr name="wnAuto" value="true" data-close></j-attr>
```

      - For Example: (Default is **true**)
        - If wn-auto is **false**.

        - If wn-auto is **true** or not declared it will return true.

    - Take Note for **Exponential Auto**:
      - Do not try to use trunc if wn-auto is **false** because your answer will be incorrect as it cuts the parts of your answer, but its possible.
      - If wn-auto is **false** then you try to round it, it will ignore the wn-auto because the unit will prioritize rounding.

- Here is the working code of using **Whole Number Auto**:

```
<p id="samplex"></p>
<jh-script>
  <j-var data="x" data-eq data-var>
    <j-convert>
      <j-attr name="for" value="convert weight" data-attr></j-attr>
      <j-attr name="unitValue" value="1" data-attr></j-attr>
      <j-attr name="type" value="milligrams" data-attr></j-attr>
      <j-attr name="target" value="metric tonnes" data-attr></j-attr>
      <j-attr name="forElement" value="#samplex" data-attr></j-attr>
      <j-attr name="tip" value="true" data-attr></j-attr>
      <j-attr name="separate" value="true" data-attr></j-attr>
      <j-attr name="unit" value="true" data-attr></j-attr>
      <j-attr name="display" value="all" data-attr></j-attr>
      <j-attr name="wnAuto" value="true" data-close></j-attr>
    </j-convert>
  </j-var>
  <j-write element="'#samplex'" method="write" output="x"
    data-write></j-write>
</jh-script>
```

- You can change the **wnAuto** from true to false

- **exp-auto** (true/false) – **Exponential Auto** if set to **true** it can converts the converted unit (that can be rounded or truncated) to exponential form.

```
<j-attr name="expAuto" value="true" data-close></j-attr>
```

  - For Example: (Default is **false**)
    - If expAuto is **false** or not declared it will return false.

    - If expAuto is **true**, you have to include the expAuto attribute

- Here is the working code of using **Exponential Auto**:

```
<p id="samplex"></p>
<jh-script>
  <j-var data="x" data-eq data-var>
    <j-convert>
      <j-attr name="for" value="convert energy" data-attr></j-attr>
      <j-attr name="unitValue" value="1" data-attr></j-attr>
      <j-attr name="type" value="electron volts" data-attr></j-attr>
      <j-attr name="target" value="joules" data-attr></j-attr>
      <j-attr name="forElement" value="#samplex" data-attr></j-attr>
      <j-attr name="tip" value="true" data-attr></j-attr>
      <j-attr name="separate" value="true" data-attr></j-attr>
      <j-attr name="unit" value="true" data-attr></j-attr>
      <j-attr name="display" value="all" data-attr></j-attr>
      <j-attr name="round" value="25" data-attr></j-attr>
      <j-attr name="expAuto" value="true" data-close></j-attr>
    </j-convert>
  </j-var>
  <j-write element="'#samplex'" method="write" output="x"
    data-write></j-write>
</jh-script>
```

  - You can change **expAuto** to **false** from true or just **remove the attribute**

❖ Forcing Attribute Modifiers

- This is where you deal with the modification of the output.
  - **fout** [add/sub(x, y)] – **Force Output** can add or subtract 1-9 to the converted output depending on the position your pointing at.

```
<j-attr name="fout" value="add(-6,1)" data-close></j-attr>
```

    - Where as **x** positions the number that you want to **add** or **subtract**, while **y** dictates the number to be added.
      - For Example: If fout = **add(-1,3)** then converted output + **3** to the tenth place of the converted output.

    - If **x** is negative, **x** will place **y** at the **decimal place value**, while **x** is positive it will place at the **whole number place value**.
      - For Example:
        - If fout = **add(-2,2)** then converted output + **2** to the hundredth place of the converted output.
        - If fout = **add(1,2)** then converted output + **2** to the ones place of the converted output.

- **fout** [add/sub(x, y)] – **Force Output**
  - You can choose a method (between **add** for adding or **sub** for subtracting the **converted output**)
    - For Example:
      - If fout = **add(-1,1)** then converted output + **1** to the tenth place of the converted output.
      - If fout = **sub(1,1)** then converted output - **1** to the ones place of the converted output.

  - **Force output** also has some rules:
    - For Example: the output is **0.324342**
      - The **whole number place value length** is **1** while the **decimal place value** length is **6**. So you should **not** put **greater** than **1** and **less** than **-6** or it will output the **default value**.
      - The only **methods** are **add** and **sub** enclosed with (x, y) like: **add(x, y)** and **sub(x ,y)**. Not following the **format** will output the **default value**.
      - In **some cases**, if the **method** is right but the **values** are **improperly implemented** the converted output will be added or subtracted to 0.

  - **For a scenario of Force Output (fout)**
    - For example:

    - The original value is **0.827180** but we want to add 1 to the last decimal place.
    - For that, specify **x** to be **negative** because we will place the number **y** at the **decimal place value**. If positive, it's the whole number place value.
    - In that case, the output is rounded at **6** so if **adding 1** at the **millionths place** (6th rightmost decimal place) of the output **x** should be **-6** and **y** should be **1.**

- **fout** [add/sub(x, y)] – **Force Output**
  - For reference:

Millions	Hundred Thousands	Ten Thousands	Thousands	Hundreds	Tens	Ones	Decimal point ↓	Tenths	Hundredths	Thousandth	Ten-Thousandths	Hundred-Thousandth	Millionths
Whole part							•	Decimal part					

  - This is the **place value chart** if your not familiar with ones, tenths, millionths and so on.

  - Here is the working code of using **Force Output**:

```
<p id="samplex"></p>
<jh-script>
  <j-var data="x" data-eq data-var>
    <j-convert>
      <j-attr name="for" value="convert data" data-attr></j-attr>
      <j-attr name="unitValue" value="1" data-attr></j-attr>
      <j-attr name="type" value="yottabyte" data-attr></j-attr>
      <j-attr name="target" value="yobibyte" data-attr></j-attr>
      <j-attr name="forElement" value="#samplex" data-attr></j-attr>
      <j-attr name="tip" value="true" data-attr></j-attr>
      <j-attr name="separate" value="true" data-attr></j-attr>
      <j-attr name="unit" value="true" data-attr></j-attr>
      <j-attr name="display" value="all" data-attr></j-attr>
      <j-attr name="round" value="6" data-attr></j-attr>
      <j-attr name="fout" value="add(-6,1)" data-close></j-attr>
    </j-convert>
  </j-var>
  <j-write element="'#samplex'" method="write" output="x"
    data-write></j-write>
</jh-script>
```

- **Note**: If there is an error, the default value will be displayed.

  -

❖ Conversion Variations

- For the **two important attributes** (method & method modifier) you can **interchange units** that you want to convert on a specific type of directive connector.

- **Volume conversion** (convert volume)

```
<p id="sampleCU"></p>
<jh-script>
  <j-var data="ueq" data-eq data-var>
    <j-convert>
      <j-attr name="for" value="convert volume" data-attr></j-attr>
      <j-attr name="unitValue" value="1" data-attr></j-attr>
      <j-attr name="type" value="liters" data-attr></j-attr>
      <j-attr name="target" value="milliliters" data-attr></j-attr>
      <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
      <j-attr name="tip" value="true" data-attr></j-attr>
      <j-attr name="separate" value="true" data-attr></j-attr>
      <j-attr name="unit" value="true" data-attr></j-attr>
      <j-attr name="display" value="all" data-attr></j-attr>
    </j-convert>
  </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

Milliliters (or **ml**)	gallons us (or **gal us**)
cubic centimeters (or **cu cm**)	cubic inches (or **cu in**)
liters (or **l**)	cubic feet (or **cu ft**)
cubic meters (or **cu m**)	cubic yards (or **cu yd**)
teaspoons us (or **tsp us**)	teaspoons uk (or **tsp uk**)
tablespoons us (or **tsbp us**)	tablespoons uk (or **tsbp uk**)
fluid ounces us (or **fl oz us**)	fluid ounces uk (or **fl oz uk**)
cups us (or **c us**)	pints uk (or **pt uk**)
pints us (or **pt us**)	quarts uk (or **qt uk**)
quarts us (or **qt us**)	gallons uk (or **gal uk**)

- **Length conversion** (convert length)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert length" data-attr></j-attr>
     <j-attr name="unitValue" value="1" data-attr></j-attr>
     <j-attr name="type" value="kilometers" data-attr></j-attr>
     <j-attr name="target" value="meters" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

nanometers (or **nm**)	inches (or **in**)
millimeters (or **mm**)	feet (or **ft**)
centimeters (or **cm**)	yards (or **yd**)
meters (or **m**)	miles (or **mi**/mil)
kilometers (or **km**)	nautical miles (or **nmi**)
microns (or **mum**, micrometer, micrometers and micrometres)	

- **Weight and Mass conversion** (convert weight)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert weight" data-attr></j-attr>
     <j-attr name="unitValue" value="1" data-attr></j-attr>
     <j-attr name="type" value="kilograms" data-attr></j-attr>
     <j-attr name="target" value="carats" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

carats (or **ct**)	grams (or **g**)	kilograms (or **kg**)
milligrams (or **mg**)	ounces (or **oz**)	
centigrams (or **cg**)	pounds (or **lb**)	
decigrams (or **dg**)	stone (or **st**)	
dekagrams (or **dag**)	short tones us (or **st us**)	
Hectograms (or **hg**)	long tones uk (or **lt uk**)	
metric tonnes (or **mt**, metric ton and metric tons)		

- **Temperature conversion** (convert temp)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert temp" data-attr></j-attr>
     <j-attr name="unitValue" value="98" data-attr></j-attr>
     <j-attr name="type" value="fahrenheit" data-attr></j-attr>
     <j-attr name="target" value="celsius" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   <j-attr name="round" value="2" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

celsius (or **c**)	fahrenheit (or **f**)	kelvin (or **k**)

- **Energy conversion** (convert energy)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert energy" data-attr></j-attr>
     <j-attr name="unitValue" value="1" data-attr></j-attr>
     <j-attr name="type" value="electron volts" data-attr></j-attr>
     <j-attr name="target" value="joules" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   <j-attr name="round" value="25" data-attr></j-attr>
   <j-attr name="expAuto" value="true" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

electron volts (or **eV**)	thermal calories (or **th cal**)
joules (or **J**/j)	food calories (or **f cal**)
kilojoules (or **kJ**)	foot pounds (or **ft lb**)
british thermal units (or **BTU**/btu)	

  - Use SI units if you prefer to shorten your entry to avoid writing the whole word of the unit.

- **Area conversion** (convert area)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert area" data-attr></j-attr>
     <j-attr name="unitValue" value="1" data-attr></j-attr>
     <j-attr name="type" value="hectares" data-attr></j-attr>
     <j-attr name="target" value="square meters" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

square millimeters (or **sq mm**)	square inches (or **sq in**)
square centimeters (or **sq cm**)	square feet (or **sq ft**)
square meters (or **sq m**)	square yards (or **sq yd**)
hectares (or **ha**)	acres (or **ac**)
square kilometers (or **sq km**)	square miles (or **sq mi**/mil)

  - SI units are **not case sensitive** and **adaptable**.
    - For example: When entering **square centimeters**, there are possible entries you could use like **sq cm**, **square cm**, **sq centimeters**, whatever those 3 categories you put even its in alternating caps it is still recognizable.
    - However, without spaces, for example like **sqcm** is **invalid** so keep in mind to put spaces if it is required.

- **Speed conversion** (convert speed)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert speed" data-attr></j-attr>
     <j-attr name="unitValue" value="62.14268" data-attr></j-attr>
     <j-attr name="type" value="mph" data-attr></j-attr>
     <j-attr name="target" value="kph" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   <j-attr name="round" value="0" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

centimeters per second (or cps/**cmps**)	miles per hour (or **mph**)
meters per second (or **mps**)	knots (or **kt**)
kilometers per hour (or **kph**)	mach (or **ma**)
feet per second (or fps/**ftps**)	

- **Time conversion** (convert time)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert time" data-attr></j-attr>
     <j-attr name="unitValue" value="1" data-attr></j-attr>
     <j-attr name="type" value="years" data-attr></j-attr>
     <j-attr name="target" value="months" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

microseconds (or **mus**)	minutes (or min)	weeks (or **wk**)
milliseconds (or **ms**)	hours (or h/**hr**)	months (or mo/**mon**)
seconds (or s/**sec**)	days (or **d**)	years (or **yr**)

  - You can also enter the type or target **without s**.
    - For example: A type or target of year is equal to years.

- **Power conversion** (convert power)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert power" data-attr></j-attr>
     <j-attr name="unitValue" value="1" data-attr></j-attr>
     <j-attr name="type" value="kilowatts" data-attr></j-attr>
     <j-attr name="target" value="watts" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

watts (or **w**)	kilowatts (or kw/**kW**)
horsepower us (or horsepower/**hp**)	
foot-pounds per minute (or **ft-lb/min** || ft-lb/minute)	
btu per minute (or BTUs per minute || BTUs per min || BTUs/min || BTUs/minute || BTU/minute || **BTU/min**)	

- **Pressure conversion** (convert pressure)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert pressure" data-attr></j-attr>
     <j-attr name="unitValue" value="1" data-attr></j-attr>
     <j-attr name="type" value="kilopascals" data-attr></j-attr>
     <j-attr name="target" value="pascals" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

atmospheres (or **atm**)	millimeters of mercury (or mmhg/**mmHg**)	
bars (or **b**)	pascals (or p/**pa**)	kilopascals (or kpa/**kPa**)
pounds per square inch (or lb-sq in || lbf-sq in || **psi**)		

- **Angle conversion** (convert angle)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert angle" data-attr></j-attr>
     <j-attr name="unitValue" value="90" data-attr></j-attr>
     <j-attr name="type" value="degrees" data-attr></j-attr>
     <j-attr name="target" value="gradians" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

Degrees (or d/**deg**)	Radians (or rad/**r**)
Gradians (or grad || gon || **g**)	

- **Basic Data conversion** (convert data)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert data" data-attr></j-attr>
     <j-attr name="unitValue" value="1000" data-attr></j-attr>
     <j-attr name="type" value="megabytes" data-attr></j-attr>
     <j-attr name="target" value="gigabytes" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- You can interchange the **type** or **target** with these units:

bits (or strictly **b**)	bytes (or strictly **B**)
kilobytes (or kb/**KB**)	megabytes (or mb/**MB**)
gigabytes (or gb/**GB**)	terabytes (or tb/**TB**)
petabytes (or pb/**PB**)	exabytes (or eb/**EB**)
zettabytes (or zb/**ZB**)	yottabytes (or yb/**YB**)

  - The table above are the **common data conversion units**.
  - However there is more data types like network data and more, see next page for more info.

- **Complex Data conversion** (convert data)

```
<p id="sampleCU"></p>
<jh-script>
 <j-var data="ueq" data-eq data-var>
   <j-convert>
     <j-attr name="for" value="convert data" data-attr></j-attr>
     <j-attr name="unitValue" value="10000" data-attr></j-attr>
     <j-attr name="type" value="megabit" data-attr></j-attr>
     <j-attr name="target" value="gigabit" data-attr></j-attr>
     <j-attr name="forElement" value="#sampleCU" data-attr></j-attr>
     <j-attr name="tip" value="true" data-attr></j-attr>
     <j-attr name="separate" value="true" data-attr></j-attr>
     <j-attr name="unit" value="true" data-attr></j-attr>
     <j-attr name="display" value="all" data-attr></j-attr>
   </j-convert>
 </j-var>
  <j-write element="'#sampleCU'" method="write" output="ueq"
    data-write></j-write>
</jh-script>
```

- There are other **not common data types** that you can **interchange** with:

kilobits (or **kbit**)	megabits (or **mbit**)
kibibits (or **kibit**)	mebibits (or **mibit**)
kibibytes (or kib/**KiB**)	mebibytes (or mib/**MiB**)
gigabits (or **gbit**)	terabits (or **tbit**)
gibibits (or **gibit**)	tebibits (or **tibit**)
gibibytes (or gib/**GiB**)	tebibytes (or tib/**TiB**)
petabits (or **pbit**)	exabits (or **ebit**)
pebibits (or **pibit**)	exbibits (or **eibit**)
pebibytes (or pib/**PiB**)	exbibytes (or eib/**EiB**)
zettabits (or **zbit**)	yottabit (or **ybit**)
zebibits (or **zibit**)	yobibits (or **yibit**)
zebibytes (or zib/**ZiB**)	yobibytes (or yib/**YiB**)

  - For the full data conversion list, enter "**convert data list**" in the for attribute then remove "list" after using it.

- For all the data conversion units, you can also enter the type or target **without s**.
  - For example: A type or target of megabyte is equal to megabytes.

- **Currency**
  - **Foundation**
    - Specialized for currency rates (using the latest exchange rate) which means it needs internet connection with only **2 methods** including the following:

currency exchange	currency list

- **Dependency Setup**: (Include the following in the HTML body)

```
<jh-dep>
    <!-- IMPORT DEPENDENCY -->
    <j-imp name="jaihoe_currency" src="./libraries"></j-imp>
</jh-dep>
```

- **Script Setup**: (Then include the following below the dependency block intended only for currency exchange)

```
<p id="sampleCE"></p>
<jh-script>
  <j-var data="x" data-eq data-var>
    <j-currency>
       <!-- INSERT PROPERTIES HERE -->
    </j-currency>
  </j-var>
  <j-write element="'#sampleCE'" method="write" output="x"
    data-write></j-write>
</jh-script>
```

  - **Currency List**
    - If you don't know what to put on the **type** or **target**, you can simply enter the following:

```
<p id="sampleCE"></p>
<jh-script>
  <j-currency>
    <j-attr name="for" value="currency list" data-attr></j-attr>
    <j-attr name="target" value="#sampleCE" data-attr></j-attr>
  </j-currency>
</jh-script>
```

- **Currency List Attribute Modifiers**
  - This method has also 2 **attribute modifiers**.
    - **display** (name/co-rate) – **Style** of the currency list name.

```
<j-attr name="display" value="co-rate" data-attr></j-attr>
```

      - For Example: (Default is **name**)
      - (Referring to **United States Currency Name**)
        - If **display**="name", it will show "United States Dollar";
        - If **display**="co-rate", it will show "United States - Dollar";
      - Along with other countries currency name!

    - **with** (iso/symbol) – **Style** of the currency list name.

```
<j-attr name="with" value="symbol" data-attr></j-attr>
```

      - For Example: (Default is **iso**)
      - (Referring to **Philippines Currency Sign**)
        - If **with**="iso", it will show "(PHP)"; //The ISO code.
        - If **with**="symbol", it will show "**(₱)**"; //The sign exactly.
      - Along with other countries currency sign!

- **Preview (Currency Exchange)**

  - 500 USD is converted to PHP which is 27,311.52 pesos
  - Take note:
    - This needs internet connection for it to work
    - Results may differ in the future, it might change

- **Working Code (Currency Exchange)**
  - Here is the working code of using **currency exchange**:

```
<p id="sampleCE"></p>
<jh-script>
  <j-currency>
    <j-attr name="for" value="currency exchange" data-attr></j-attr>
    <j-attr name="target" value="#sampleCE" data-attr></j-attr>

    <j-attr name="amount" value="500" data-attr></j-attr>
    <j-attr name="from" value="usd" data-attr></j-attr>
    <j-attr name="to" value="php" data-attr></j-attr>

    <j-attr name="sign" value="symbol" data-attr></j-attr>
    <j-attr name="tip" value="iso" data-attr></j-attr>

    <j-attr name="round" value="2" data-close></j-attr>
  </j-currency>
</jh-script>
```

- **Required Attribute Modifiers**
  - **Amount** – This is the currency amount to be converted

```
<j-attr name="amount" value="500" data-attr></j-attr>
```

  - **From** – This is the currency type to be converted

```
<j-attr name="from" value="usd" data-attr></j-attr>
```

  - **To** – This is the currency type to be output along with the amount

```
<j-attr name="to" value="php" data-attr></j-attr>
```

- **Basic Attribute Modifiers**
  - This directive connector also has 7 other method modifiers.
    - **display** (all/converted) – **Display** style of the currency rate.

```
<j-attr name="display" value="all" data-attr></j-attr>
```

    - For Example: (Default is **converted**)
      - **1 USD converting to PHP**
      - If **true**, it will show "$1 = ₱54.62"; (sign is **symbol**)
      - If **false**, it will show "₱54.62"; (sign is **symbol**)

- **Basic Attribute Modifiers**
  - **separate** (true/false) – **Separates** number of the currency rate.

```
<j-attr name="separate" value="false" data-attr></j-attr>
```

    - For Example: (Default is **false**)
      - **100 USD converting to PHP**
        - If **true**, it will show "$100 = ₱5,462.30"; (display is **all**, sign is **symbol**)
        - If **false**, it will show "$100 = ₱5462.30"; (display is **all**, sign is **symbol**)

  - **sign-space** (true/false) – Put **spaces** between the sign and value of the currency rate depending on where the sign is located.

```
<j-attr name="signSpace" value="true" data-attr></j-attr>
```

    - For Example: (Default is **false**)
      - **100 USD converting to PLN**
        - If **true**, it will show "$ 100 = 434.93 zł"; (display is **all**, sign is **symbol**)
        - If **false**, it will show "$100 = 434.93zł"; (display is **all**, sign is **symbol**)

  - **sign** (false/symbol/iso/name/co-rate) – Put **sign** before or after currency value depending on the countries banknote style.
    - For Example: (Default is **false**)
      - **1 USD converting to PHP**
      - If sign="**false**", it will show "1 = 50.6"; (display is **all**)
      - If sign="**symbol**", it will show "$1 = ₱50.6"; (display is **all**)
      - If sign="**iso**", it will show "1 USD = 50.6 PHP"; (display is **all**)
      - If sign="**name**", it will show:
        - "1 United States Dollar = 50.56 Philippine Peso"; (display is **all**)
      - If sign="**co-rate**", it will show:
        - "1 United States - Dollar = 50.56 Philippines – Peso"; (display is **all**)

- **Basic Attribute Modifiers**
  - **tip** (false/symbol/iso/name/co-rate) – Additional accessory for instance sign is symbol when you **hover** the **currency value** you can set a tooltip for the information of that currency value.

```
<j-attr name="tip" value="iso" data-attr></j-attr>
```

  - For Example: (Default is **false** which does not display a tooltip)
    - **5 USD converting to PHP**
    - If tip="**symbol**", it will show "$5 = ₱819.35"; (display is **all**)
    - If tip="**iso**", it will show "5 USD = 819.35 PHP"; (display is **all**)
    - If tip="**name**", it will show:
      - "5 United States Dollar = 819.35 Philippine Peso"; (display is **all**)
    - If tip="**co-rate**", it will show:
      - "5 United States - Dollar = 819.35 Philippines – Peso"; (display is **all**)

- **Rounding Attribute Modifiers**
  - This is where the converted currency amount (output) can be rounded.
    - **round** (number input) – Rounds the converted currency rate's decimal number raging from 0 to 2 only.

```
<j-attr name="round" value="2" data-close></j-attr>
```

    - For Example: **50 USD converting to PLN**

- **Rounding Attribute Modifiers**
  - **round** (number input) - **Conditions**
    - If **round** is **not defined** then the converted currency rate's is equal to 217.46565 zł
    - If **round** is "2" then the converted currency rate's is equal to 217.47zł
    - If **round** is "0" then the converted currency rate's is equal to 217zł
    - If **round** is "3" or higher its value will still be rounded to 2 which is 217.47zł
    - If **round** is "-1" or lower its value will still be rounded to 0 which is 217zł

  - **trunc** (number input) – Splits the converted currency rate's decimal number raging from 0 to 2 only.

```
<j-attr name="trunc" value="2" data-close></j-attr>
```

    - For Example: **50 USD converting to PLN**
      - If **trunc** is "2" then the converted currency rate's is equal to 217.46zł
      - If **trunc** is"0" then the converted currency rate's is equal to 217zł
      - If **trunc** is "3" or higher its value will still be truncate to 2 which is 217.46zł
      - If **trunc** is "-1" or lower its value will still be truncate to 0 which is 217zł

- **Currency Variations**
  - You can interchange the **from** or **to** with these rates:

United States Dollar (USD)	Icelandic Krona (ISK)
United Arab Emirates Dirham (AED)	Japanese Yen (JPY)
Argentine Peso (ARS)	South Korean Won (KRW)
Australian Dollar (AUD)	Kazakhstani Tenge (KZT)
Bulgarian Lev (BGN)	Maldivian Rufiyaa (MVR)
Brazilian Real (BRL)	Mexican Peso (MXN)
Bahamian Dollar (BSD)	Malaysian Ringgit (MYR)
Canadian Dollar (CAD)	Norwegian Krone (NOK)
Swiss Franc (CHF)	New Zealand Dollar (NZD)
Chilean Peso (CLP)	Panamanian Balboa (PAB)
Chinese Yuan (CNY)	Sol (PEN)
Colombian Peso (COP)	Philippine Peso (PHP)
Czech Koruna (CZK)	Pakistani Rupee (PKR)
Danish Krone (DKK)	Poland Zloty (PLN)
Dominican Peso (DOP)	Paraguayan Guarani (PYG)
Egyptian Pound (EGP)	Romanian Leu (RON)
Euro (EUR)	Russian Ruble (RUB)
Fijian Dollar (FJD)	Saudi Riyal (SAR)
Pound Sterling (GBP)	Swedish Krona (SEK)
Guatemalan Quetzal (GTQ)	Singapore Dollar (SGD)
Hong Kong Dollar (HKD)	Thai Baht (THB)
Croatian Kuna (HRK)	Turkish Lira (TRY)
Hungarian Forint (HUF)	New Taiwan dollar (TWD)
Indonesian Rupiah (IDR)	Ukrainian Hryvnia (UAH)
Israeli New Shekel (ILS)	Uruguayan Peso (UYU)
Indian Rupee (INR)	South African Rand (ZAR)

  - **Notes**:
    - The name and co-rate symbolic letters are not applicable, just use regular letters only. (For Example: Poland Złoty, change "ł" to "l" as shown in the table)
    - Additionally, currency symbols cannot be **from** or **to** attributes because of sign similarities compared to name, co-rate and iso.

- **Fun**
  - **Foundation**
    - This library makes your page interactive and lively by using these 3 methods:

fun text	fun title	fun behavior

- **Dependency Setup**: (Include the following in the HTML body)

```
<jh-dep>
    <!-- IMPORT DEPENDENCY -->
    <j-imp name="jaihoe_fun" src="./libraries"></j-imp>
</jh-dep>
```

- **Script Setup**: (Then include the following below the dependency block intended only for currency exchange)

```
<jh-script>
   <j-fun>
     <!-- INSERT PROPERTIES HERE -->
   </j-fun>
</jh-script>
```

  - **Fun Text**
    - This method is where you can make text colorful!
      - **Background Overlay Text**
        - This adds background within the text written

        - The working code is on the next page.

- **Background Overlay Text (Working Code)**
  - Here is the working code of using background overlay text:

```
<h2 id="bgtSAMPLE">It's me, hi, I'm the older, it's me! At tea time,
everybody agrees. I'll stare directly at the moon but never on
reflection. It must be exhausting, always cheering for the anti-
hero.</h2>

<jh-script>
  <j-fun>
    <j-attr name="for" value="fun text" data-attr></j-attr>
    <j-attr name="target" value="#bgtSAMPLE" data-attr></j-attr>
    <j-attr name="method" value="background overlay text"
      data-attr></j-attr>
    <j-attr name="repeat" value="no-repeat" data-attr></j-attr>
    <j-attr name="size" value="cover" data-attr></j-attr>
    <j-attr name="clip" value="text" data-attr></j-attr>
    <j-attr name="textFillColor" value="transparent"
      data-attr></j-attr>
    <j-attr name="bgColor" value="transparent" data-attr></j-attr>
    <j-attr name="bgImage" value=" https://bit.ly/jhbgspace"
      data-attr></j-attr>
    <j-attr name="bgURL" value="true" data-attr></j-attr>
  </j-fun>
</jh-script>
```

- **Blinking Text**
  - Literally makes a blinking text that blinks continuously

- **Blinking Text (Working Code)**
  - Here is the working code of blinking text:

```
<div align="center">
  <h1 id="sampleba">Breaking News!</h1>
  <h1 id="samplebb">[This just in, john break the record]</h1>
</div>
<jh-script>
  <j-fun>
    <j-attr name="for" value="fun text" data-attr></j-attr>
    <j-attr name="target" value="#sampleba" data-attr></j-attr>
    <j-attr name="method" value="blinking text" data-attr></j-attr>
    <j-attr name="speed" value="250" data-attr></j-attr>
    <j-attr name="delay" value="5" data-attr></j-attr>
    <j-attr name="canStop" value="true" data-attr></j-attr>
  </j-fun>
  <j-fun>
    <j-attr name="for" value="fun text" data-attr></j-attr>
    <j-attr name="target" value="#samplebb" data-attr></j-attr>
    <j-attr name="method" value="blinking text" data-attr></j-attr>
    <j-attr name="speed" value="500" data-attr></j-attr>
    <j-attr name="delay" value="250" data-attr></j-attr>
    <j-attr name="canStop" value="false" data-attr></j-attr>
  </j-fun>
</jh-script>
```

  - Try to double click “Breaking News!” to make it stop blinking.

- **Rainbow Text**
  - Literally makes a rainbow text that change color continuously

- **Rainbow Text (Working Code)**
  - Here is the working code of rainbow text:

```
<div align="center">
  <h1 id="samplera">Rainbow Text!</h1>
  <h1 id="samplerb">(For the Unicorns)</h1>
</div>
<jh-script>
  <j-fun>
    <j-attr name="for" value="fun text" data-attr></j-attr>
    <j-attr name="target" value="#samplera" data-attr></j-attr>
    <j-attr name="method" value="rainbow text" data-attr></j-attr>
    <j-attr name="speed" value="20" data-attr></j-attr>
  </j-fun>
  <j-fun>
    <j-attr name="for" value="fun text" data-attr></j-attr>
    <j-attr name="target" value="#samplerb" data-attr></j-attr>
    <j-attr name="method" value="rainbow text" data-attr></j-attr>
  </j-fun>
</jh-script>
```

- **Fun Title**
  - This method is where you can make the document title interactive
    - **Scrolling Title**
      - Incase your title is too long, you can make it scroll using this method

```
<h2>Scrolling Title! (check the title bar)</h2>
<jh-script>
 <j-fun>
  <j-attr name="for" value="fun title" data-attr></j-attr>
  <j-attr name="method" value="scrolling title" data-attr></j-attr>
  <j-attr name="speed" value="250" data-attr></j-attr>
  <j-attr name="spacing" value="3" data-attr></j-attr>
 </j-fun>
</jh-script>
```

- **Scrolling Title (Character Issues)**
  - If you encounter this problem:

  - Just **include** the following inside the **head** tag:

```
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
```

- **Blinking Title**
  - Makes the title bar blink, making it like news flash!

```
<h2>Blinking Title! (check the title bar)</h2>
<jh-script>
 <j-fun>
  <j-attr name="for" value="fun title" data-attr></j-attr>
  <j-attr name="method" value="blinking title" data-attr></j-attr>
  <j-attr name="speed" value="500" data-attr></j-attr>
 </j-fun>
</jh-script>
```

- **Fun Behavior**
  - This method is where you can make interactive page behavior
    - **Smooth Go to Top**
      - When you scroll to the bottom, you can scroll up smoothly

```
<div id="sentence" style="margin-bottom:30px;text-align:
justify;text-justify: inter-word;">
</div>
<div align="center" style="margin-bottom: 30px;">
  <button style="font-size:18px;padding:10px;" id="gttbtnx">
    Go to top?</button>
</div>
<jh-script>
  <j-write element="'#sentence'" data-write-block>
    <j-random type="sentence" param="150" data-random></j-random>
  </j-write>
  <!-- Making Smooth Go to Top -->
  <j-fun>
   <j-attr name="for" value="fun behavior" data-attr></j-attr>
   <j-attr name="target" value="#gttbtnx" data-attr></j-attr>
   <j-attr name="method" value="smooth gtt" data-attr></j-attr>
  </j-fun>
</jh-script>
```

      - Just **click** the “Go to top?” **button** and the page will go up

- **To Attract**
  - A click event which makes the object follow the cursor
  - For example: (A follow button)

```
<div align="center">
 <h1 id="sampleSE">Social Experiment</h1>
 <p>(Can you follow me? I will follow you back.)</p>
</div>

<div align="center" style="margin-bottom: 30px;">
 <button style="font-size:18px;padding:10px;"
   id="followYouBack">Follow</button>
 <button style="font-size:18px;padding:10px;display:none;"
   class="followerAx">Follower</button>
</div>

<jh-script>
  <!-- Making a Follow Button -->
   <j-fun>
    <j-attr name="for" value="fun behavior" data-attr>
      </j-attr>
    <j-attr name="target" value="#followYouBack" data-attr>
      </j-attr>
    <j-attr name="method" value="that attract" data-attr>
      </j-attr>
    <j-attr name="attract" value="#followerAx" data-attr>
      </j-attr>
   </j-fun>
</jh-script>
```

  - When you click the button, it will literally follow the cursor
  - When you click it again, it will stay to its original position

- **To Repel**
  - A mouse event that makes an object avoid the mouse

  - Working code not available, try to check our website for updates.
    - https://jaihoejs.github.io/